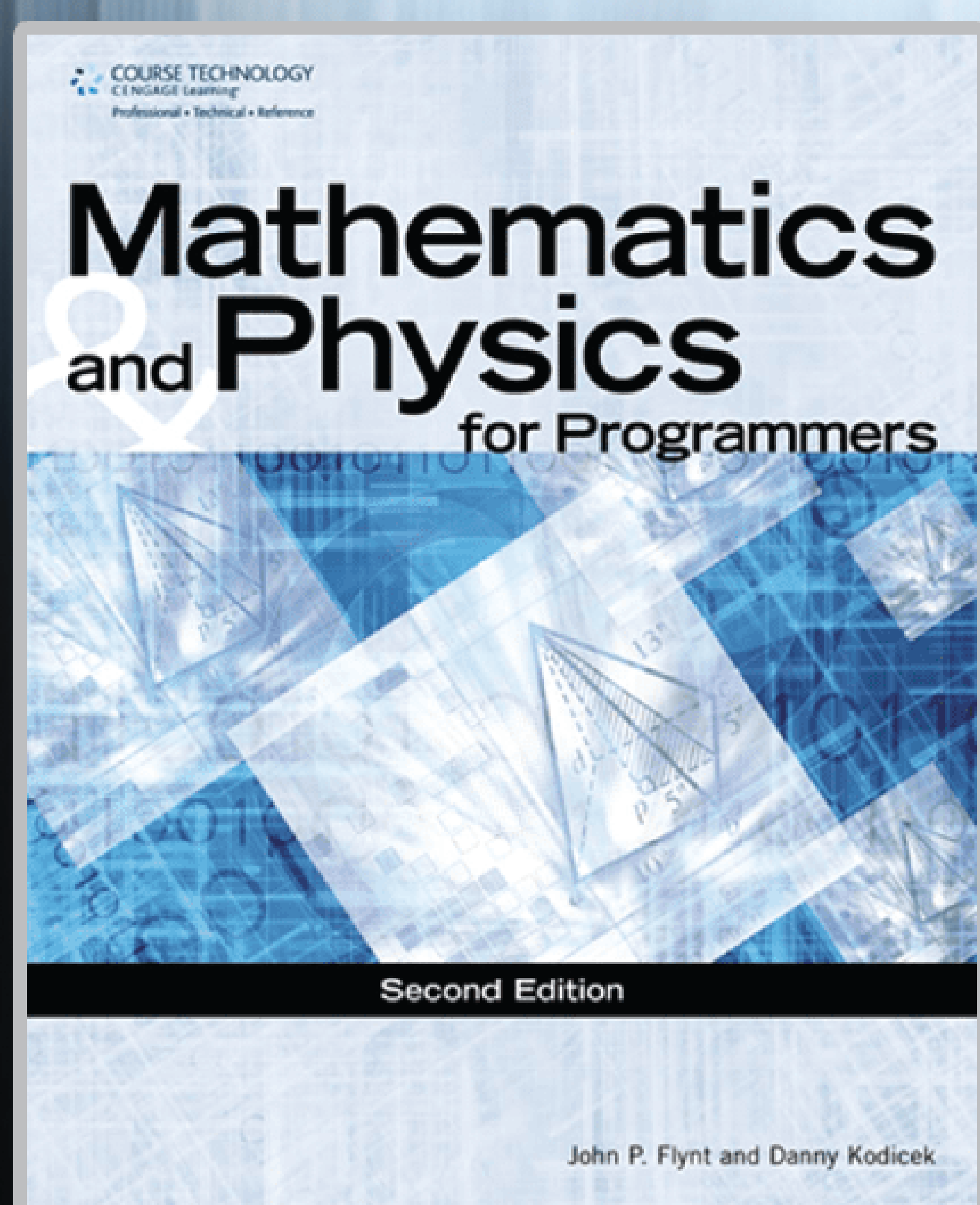
CENGAGE Learning™

游戏中的数学与物理学

（第2版）

[美]John Patrick Flynt
Danny Kodicek 著
周建娟 译

Mathematics and Physics for Programmers,Second Edition



清华大学出版社

游戏中的数学与物理学

（第 2 版）

[美] John Patrick Flynt

Danny Kodicek　著

周建娟　译

清华大学出版社

北　京

内 容 简 介

本书详细阐述了与游戏数学和物理学相关的基本解决方案，主要包括数字，数学运算，代数运算，几何学和三角学，向量，微积分，加速度、质量和能量，简单形状之间的碰撞检测，碰撞处理方案，摩擦力，绳索、滑轮和传送带，振荡现象，3D 形状，转换操作，碰撞检测，光照和纹理，建模技术，加速方案，贴图游戏，迷宫类游戏，博弈论和人工智能，搜索技术等内容。此外，本书还提供了相应的示例、伪代码，以帮助读者进一步理解相关方案的实现过程。

本书既适合作为高等院校计算机及相关专业的教材和教学参考书，也可作为相关开发人员的自学教材和参考手册。

北京市版权局著作权合同登记号　图字：01-2014-8032
Mathematics and Physics for Programmers,Second Edition
John Patrick Flynt, Danny Kodicek 著 周建娟 译

978-7-302-37951-5

Cengage Learning Asia Pte. Ltd.
5 Shenton Way, # 01-01 UIC Building, Singapore 068808

图书在版编目（CIP）数据

游戏中的数学与物理学：第 2 版/（美）弗林特（Flynt,J.P.），（美）科迪塞克（Kodicek,D.）著；周建娟译．—北京：清华大学出版社，2014
书名原文：Mathematics and Physics for Programmers,Second Edition
ISBN 978-7-302-37951-5

I. ①游…　II. ①弗…　②科…　③周…　III. ①游戏程序–程序设计　IV. ①TP311.5

中国版本图书馆 CIP 数据核字（2014）第 207864 号

责任编辑：钟志芳
封面设计：刘　超
版式设计：文森时代
责任校对：王　云
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：203mm×260mm　**印　　张：**26.75　**字　　数：**696 千字
版　　次：2014 年 12 月第 1 版　**印　　次：**2014 年 12 月第 1 次印刷
印　　数：1～4000
定　　价：89.00 元

产品编号：055783-01

译 者 序

数学和物理学可视为一类实践项目，若非亲自解决某一问题，通常难以理解其真实含义，这一理念对于游戏开发尤其如此。

针对这一问题，本书详细阐述了游戏中的数学和物理设计方案，涵盖了丰富的内容，主要包括数字，数学运算，代数运算，几何学和三角学，向量，微积分，加速度、质量和能量，简单形状之间的碰撞检测，碰撞处理方案，摩擦力，绳索、滑轮和传送带，振荡现象，3D 形状，转换操作，碰撞检测，光照和纹理，建模技术，加速方案，贴图游戏，迷宫类游戏，博弈论和人工智能，搜索技术等内容。值得一提的是，本书并非一本纯理论书籍，除了对相关内容进行全面、系统的讲解以外，其设计思想、数据结构和算法均辅以对应的代码示例，以帮助读者进一步理解计算方案的实现过程。

在翻译过程中译者时刻感觉到责任重大，唯恐出现任何错误愧对原著和读者。但是，由于技术和英文语言理解方面的水平所限，书中难免会存在一些错误和不当之处，敬请读者批评指正，我们将不胜感激。希望本译著对进一步推动我国的游戏产业相关领域的研究与应用起到推动作用。

在本书的翻译过程中，除周建娟之外，张欢欢、李莉、米玥、吴帆、朱琳琳、王梅、潘冰玉、赵雷、李海俊、程聪、王巍、孙健、皮雄飞、李保金、史云龙、刘凌、朱利平、胡子文、王典、李中卉、丁妍、孙年果、姚楷峰、李亚楠、梁洪娇、王晓晓也参与了本书的翻译工作，在此一并表示感谢。

译　　者

前　　言

本书适用读者

本书适用于不同层次的读者，并涵盖了数学、物理学以及生物学等内容，以帮助程序员（特别是游戏程序员）理解此类内容的程序编写方式。即使读者关注的重点并非是游戏设计，相信也可从本书中获取有价值的信息。

如果读者熟悉某种程序设计语言，例如，Java、JavaScript、Python、ActionScript、Lingo、C/C++、Visual Basic、Perl 或 C#，则阅读效果定会得到较大的改观，甚至可扩展至 MATLAB、Mathematica 或 Maple 等其他脚本语言。另外，本书代码示例均采用伪代码进行描述，因而相关内容适用于任何语言。

本书主要讨论数学、物理学等知识于程序设计的应用方式，尤其是游戏环境。当然，相关内容同样适用于其他领域。如果读者对以下问题感兴趣，则有必要认真阅读本书：基于神经元集合或遗传基因学的应用程序如何应用于游戏中？迷宫理论以及细分操作如何执行？线性代数的含义以及如何使用向量和矩阵构建并控制图形对象？如何运用基本的运算知识并在高级程序设计环境中与 3D 对象协同工作？另外，读者也可对相关知识进行回顾。

本书内容

顾名思义，本书针对程序设计讨论数学和物理方面的知识。需要说明的是，多数时候，相关内容并不会逐步予以介绍，读者可复制伪代码示例，经过适当调整后用于自己的项目中。本书编写的主要目的在于讨论游戏环境下数学和物理处理方案，进而从广度和深度方面重新认识某些问题。

本书内容可划分为 4 类，如下所示：

- 基本内容。此类内容多为日常生活所面临的话题，并反映出数学中的基本概念，其中包括数字、代数和几何学的基本原理。这一类话题多出现于本书的第 1 部分以及第 2、3、4 部分中的前几章中（涉及物理学和 3D 数学）。本书将对基本内容予以重点讲解，并包含相关推导过程和示例代码。
- 高级话题。此类话题源自数学和物理学的高级领域，并提供了某一话题的完整讲解，进而求解相关问题，而非日常见到的基础知识，其中包括：积分学（第 6 章）、高级物理概念（第 3 部分）以及 3D 数学，特别是与 3D 渲染器相关的内容（第 4 部分）。高级话题穿插于本书讲解过程中，包括原理和术语的解释、少量的示例以及某些重要结果的推导过程。

- 应用话题。相关内容考察基本概念于复杂环境下的应用方式，且主要分布于本书的第 2 部分以及第 3 部分和第 4 部分的某些章节中。应用话题主要讨论某一话题的特定示例并对其进行深入讨论。同时辅以详细的代码示例以及数学结果的推导过程。其中，对应示例有助于读者理解其他类似问题，进而采用不同的方法求解同一问题。
- 扩展话题。该话题涉及某些高级领域，并以此接触更为广泛的内容。扩展话题采用简单扼要的方式进行介绍，且出现于本书的第 3、4、5 部分中，其目的在于使读者了解某些基本概念和术语，且无须过分关注细节内容。对应的完整讨论则超出了本书的范围，读者可在此基础上阅读相关书籍以获取更多内容。高级话题涉及的章节包含了少量的代码示例和计算方程，以及大量的文字解析和图像示例。

当然，上述分类方式在一定程度上存在交集，各章均包含彼此交叠的内容。本书的主要目标是理解程序设计中的数学原理和概念，因而并不会事无巨细地对每个专题进行讲解。这里，希望读者深入理解相关概念，并在遇到新问题时可做到有的放矢。据此，本书涵盖了数学和物理学原理、数学计算示例以及通用概念和术语。

各章结尾提供了相应的编程练习，另外，本书合作网站也包含了大量的代码示例。多数时候，对应示例可在章节文件中找到，但某些较为冗长的示例则需要通过特定的名称得到，全部内容均在 HTML 文件中进行定位，以方便、直观地对示例予以访问。

如何阅读本书

本书采用内容“累积”方式进行讲解，多数章节均会参考前述所讨论的知识。因此，建议读者先期快速浏览本书全部内容。即使读者对某些数学和物理学内容十分熟悉，但有时依然无法面面俱到。

数学可视为一类实践项目，若非亲自解决某一问题，通常难以理解其真实含义。对此，读者应完成每章结尾给出的练习题。实际上，相关练习题并非独立存在，读者可思考如何将其整合至大型项目中，尝试将所学原理应用于游戏中，并整合新元素以改善现有游戏项目。作为一名程序员，应通过这一方式掌握所学知识背后的原理。

人们通常对新生事物存在某种“恐惧感”，即使之前曾完美地解决了某些复杂任务。若读者难以理解相关内容，则可尝试再次阅读。若问题仍未解决，则可分析其中的原因——是否未曾理解相关术语？对此，读者可对相关知识进行适当回顾、阅读有关章节、考察示例、绘制图表并对其进行重新描述。如果全部工作均无效，则可暂时放弃该问题，以期待后续内容是否可提供相关线索。

关于伪代码

本书目标旨在引入必要的数学知识，且并未介绍相关的编程风格。此处，读者无须纠结于程序语言的编程规范、实现方式、用户接口，相应地，本书较少涉及变量、对象、数组以及精灵（sprite）对象。相反，读者可关注执行相关任务的函数和算法。本书涵盖了丰富的数学特征，并可方便地

在任何语言中予以实现，其具体过程可视为一个黑盒。另外，通过适当整合（与代码其他部分）以及程序设计语言中的简便方法，相同的数学操作可更好、更快地予以实现。实际上，这一情况时有发生。

为了有效地避免编程风格的干扰，本书全部示例均采用伪代码表示，进而可将使用不同语言的程序员区分开来。本书代码关注细节问题，且采用易读方式加以编写。从本质上讲，伪代码针对解释程序加以编写，而非计算机设备。

少数时候，代码的部分内容予以省略，进而着重强调某些重要问题。其中，省略内容采用“||”标记，代码中的注释则采用“//”标记。

本书辅助网站中的大多数代码采用 Lingo 语言编写，作为一类非编译型语言，该高级语言采用无类型变量，且兼具灵活性，适用于面向对象以及过程式语言。

本书尽量避免与变量、数组等内容相关的规范，尽管此类内容在某些时候较为重要。本书采用 Lingo 语言风格，例如，数组的首个元素定义为 A[1]而非 A[0]。

如何定位代码示例

如前所述，代码示例采用伪代码编写，因而无法直接将其复制、粘贴至编译器中并运行相应程序。

大多数代码示例用 HTML 文档表示，并通过下列两种方式进行组织：

- 章节文件夹。若代码仅与某一章的主题相关，则可找到与该章具有相同名称的文件夹，进而访问一个或多个对应的 HTML 文件。多数时候，读者将会得到与该章具有相同名称的文件，例如 Chapter 7.html。有些时候，读者则会搜索具有不同名称的文件，对此，可通过浏览器对其进行定位，进而获取与当前章节相关的程序。
- 主索引。该方案更为有效且予以优先推荐。对此，可在名为 Indexed Programs 的文件夹中访问 index.html 文件。该文件可帮助读者访问特定的章节主题，并对全书的不同主题进行有效的组织。通过这一方式，读者可深入理解特定主题，并了解相关内容的组织方式。

当首次打开 index.html 文件时，对应内容可能稍显空泛，对此，下列内容提供了相关链接：

- Castlib tileScroller。该链接与本书最后几章所讨论的程序有关，其中包括贴图（第 23 章）以及搜索和优化方案。
- Castlib math。该链接与本书前 18 章讨论的程序相关，并涵盖了其中的数学运算。
- Castlib mazes。该链接与本书第 23～25 章讨论的程序相关，并用于处理迷宫和 AI 问题。
- Castlib pool。该链接包含了与撞球游戏相关的文件。该游戏首次出现于第 11 章，并在后续章节中多次被引用。
- Castlib simple3D。该链接提供了与 3D 图形相关的程序。对应程序首次出现于第 17 章，后续章节则在不同环境下多次对其加以引用。

需要注意的是，Castlib 表示分组项。其中，Cast 引用了与普通任务相关的、游戏中的对象组。也就是说，示例代码根据相关主题进行分组。另外，在查看过程中，读者可能习惯于打开 index.html 文件，但在一段时间后，读者还可能根据单词项访问示例代码。

错误、遗漏和意见反馈

鉴于本书的篇幅和讨论范围，错漏之处在所难免。对此，读者可向出版社提供相应的意见反馈，我们将会在第一时间对其进行处理。

本书辅助网站

读者可访问 www.courseptr.com/downloads 下载代码示例文件，需要说明的是，该网址将直接转向 Cengage Learning 网站。

本书作者

作为本书作者和多部书籍的联合作者，John Flynt 的作品涉及程序设计、数学、软件工程和游戏开发，其中包括 *Software Engineering for Game Developers*、*In the Mind of a Game*、*Perl Power!: The Comprehensive Guide*、*Java Programming for the Absolute Beginner*，*Second Edition*、*Beginning Math Concepts for Game Developers* 以及 *Java ME Game Programming*，*Second Edition*。

Danny Kodicek 就职于英国的 Sunflower Learning 公司，负责科学仿真以及相关工具的研发。他所开发的软件已被翻译为 15 种语言，并在世界范围内广泛销售。他的另一个身份是一名自由作家，其客户包括 BBC 和英国皇家空军。除此之外，他也是获奖网站 TimeHunt 的联合开发者。

目　　录

第 1 部分　数 学 知 识

第 2 部分 物理学基本内容

第 3 部分　复 杂 运 动

第 4 部分　3D 数学

第 5 部分　游 戏 算 法

第 1 部分　数学知识

本书第 1 部分讨论基本的数学概念，相信大多数读者对此不会感到陌生。本章内容十分重要，以使读者在后续学习过程中可处理更为复杂的概念。

第 1 章　数　字

本章包含如下内容：

- 概述。
- 数字的书写方式。
- 计算机的数字表达方式。
- 标准数字和计算数字。
- 常见函数。
- 舍入误差和性能。

1.1　概　述

数字是数学的基础，相信大多数读者也持有这一观点。该结论源自现代科学观念，最初，人们只是通过几何形状记录数字。同样，数字也是计算机科学的基础内容之一，因而深入理解数字及其工作方式对于程序设计而言十分重要。

后续章节将介绍数字于计算机的表达方式，以及基于计算机的处理方法，并着重强调数字的书写表达方式以及计算机表达方式之间的差异。

【提示】本书合作网站中包含了第 1 章中的示例代码，对应位置为 www.courseptr.com/downloads 中的 Chapter 1 文件夹。

1.2　数字的书写方式

数字的发展过程并非一蹴而成，今天所看到的数字（如正数、负数、0、整数、有理数、无理数以及复数）已历经数千年的演化过程。本章并不打算回顾这段历史，但与通用定义相关的某些要点依然具有实际意义。

1.2.1　整数、有理数和无理数

数学家们将数字世界划分为多个子集，下面首先介绍自然数。自然数通过 N 符号表示，即孩子们所学的 1，2，3，4 等数字。除此之外，自然数中还包括数字 0。其中，各数字均表达了

一项“独立”事物。当执行负数的加法运算时，数据域则涉及整数，并通过符号 ℤ 表示。

何为负数？读者可通过下列交易活动加深对负数的理解。假设读者手中持有 5 枚弹珠，Anne 手中持有 4 枚弹珠，若读者给 Anne 两枚弹珠，则 Anne 将“得到”两枚弹珠，或者说，Anne 给予了读者−2 枚弹珠。无论如何，读者均向一方添加了两枚弹珠，而从己方减去了两弹枚珠。

相应地，可将上述行为扩展至乘法运算。若采用“反向”操作考察负数，则可知晓两个负数的乘积结果为正数。假设读者给予 Anne 两枚弹珠，即 Anne 给予读者−2 枚弹珠。若读者执行该交易−2 次，反之，Anne 则执行该操作 2 次。因此，若根据−4 加以反向考察，则最终结果为+4。

上述过程多少令人费解，因此，几个世纪以来，人们难以接受负数这一概念。小数（即两个整数的商）则很快为人们所接受，首先，存在各种 1 的商（小数），例如 1/2 和 1/3。同时，当 2/1 变为 1/2 时，将产生自然数的倒数这一概念，随后是普通分数，即商值大于 1，例如 5/4。此时产生了有理数集，其符号表示为ℚ，并包含了正小数和负小数以及整数。由于整数可视为分母为 1 的特殊小数，因此也被归类于有理数中。

【提示】当两个数字执行除法运算时，即可得到“商”这一形式。若两个整数相除，则可表示为分数形式。其中，上方数字为分子，下方数字为分母。某些时候，商值也意味着除法的整数部分，稍后将对此加以讨论。类似地，和、差、积分别表示为两个数字的加法、减法以及乘法运算。

1.2.2　无理数和实数

除了有理数之外，还存在一类无理数，此类数字无法通过两个整数的商加以表示。长久以来，人们并不认为无理数真实存在。毕达哥拉斯认为，至少存在一个数（即 2 的平方根）可定义为无理数。自此（约 550 B.C.E），随着视野不断地开阔，数学家们逐渐认识到，无理数的数量大于有理数的数量。

有理数和无理数构成了实数集，并通过符号 ℝ 表示。另外，无理数还涵盖了虚数 i，即−1 的平方根。无理数的相关内容超出了本章的讨论范围，在第 8 章介绍四元数时，将对其予以深入讨论。

【提示】在数学方式中，可通过 $\sqrt{n}$ 表达数字 n 的平方根。在程序设计语言中，平方根则通过函数予以描述，例如 sqrt(n)。若 n 的平方根结果为 m，则有 $m \times m = n$。当通过该方式对数字自身执行此项操作时，则对应结果称作平方值，记为 n^2、n^2 或 power(n,2)。需要说明的是，平方仅是幂或指数形式的一个特例。若 p 表示为正数，则 n 的 p 次方表示为 n^p 或 n^p，即数字 n 自身相乘 p 次。由于两个负数相乘后结果为正值，因而各正数包含两个平方根，即正平方根和负平方根，而负数则不包含实根（稍后将对负数 p 的处理过程加以分析）。

1.2.3　数位串形式的数字

数字可通过多种方式表示。对于数字 5 而言，可采用几何方式对其加以描述，例如五角星，

也可采用包含 5 个点的点集对其进行定义。除此之外，还可通过物理方式表现数字，例如算盘上的 5 个算珠。当然，还存在其他方案并根据各自含义表达符号“5”，各表达方式均具有自身的优势。例如，若采用算珠方式，则可简单地来回移动算珠执行加法和减法运算。相比较而言，若数值较大，则符号方式具有明显的优势。

实际上，各数字均无法采用唯一的符号表示其数值，相反，可通过有限的符号集并经适当整合后表示既定数字，且无须担心数值的大小。这里，可使用进制方案表达数字，例如，可采用数字 b 作为进制。当定义进制 b 中的某一数字 n 时，需要使用递归算法表示具体数字，对应伪代码如下所示：

```
function numberToBaseString(Number, Base)
  if Number is less than Base then set Output to String (Number)
  otherwise
   || Find the remainder Rem when you divide Number by Base
   set Output to Rem
   set ReducedNumber to (Number - Rem) / Base
   set RestOfString to numberToBaseString (ReducedNumber, Base)
   append RestOfString to the front of Output
   end if
  return Output
end function
```

【提示】在 numberToBaseString()函数中，当使用 n 除以 m 时，余数表示为最小数字 r，且对于整数 a 而言，满足 $m \times a + r = n$。相应地，由于 $7 = 3 \times 2 + 1$，因而 7 除以 3 时的余数为 1。第 2 章将深入考察两个数字相除时的余数计算过程。

numberToBaseString()函数使用了递归算法，顾名思义，该函数将数字转换为字符串。numberToBaseString()函数包含两个参数 Number 和 Base。若采用 354 作为 Number 的参数，10 作为 Base 的参数，则 354 除以 10 的余数为 4，该值表示为 354 中的个位。作为余数，数字 4 写入至输出内容中。随后，减去余数后的结果为 350，该值再次与 10 执行除法运算，且对应结果为 35，并再次输入至当前算法中，最终结果为 5，即 354 中的十位。再次执行循环迭代计算，其结果为 3，即 354 中的百位。经反向整合后，最终将生成字符串“354”，即函数中的首个参数。尽管上述过程并非必需，但该过程的重点在于以递归方式考察数字 354 和十进制之间的关系（附录 A 提供了这一话题的其他内容）。

类似地，若采用逆操作，也就是说，使用字符串作为参数，则函数最终结果为数值，对应的 baseStringToValue()函数如下所示：

```
function baseStringToValue(DigitString, Base)
  if DigitString is empty then set Output to 0
    otherwise
    set Output to the last digit of DigitString
    set RemainingString to all but the last digit of DigitString
    set ValueOfRemainingString to
      baseStringToValue (RemainingString, Base))
    add Base * ValueOfRemainingString to Output
  end if
```

```
  return output
end function
```

当采用进制方式书写数字时，需对数字各位指定进制的位置。考察数字 2631，其中，首个数字是 1，即该数字乘以 10 的 0 次方；第 2 个数字 30 则表示 10 的 1 次方乘以 3；第 3 个数字 600 表示 10 的 2 次方乘以 6，该过程持续进行。当采用加法序列操作时，可得到如下形式的表达式：

$$2\times1000 + 6\times100 + 3\times10 + 1\times1$$

1.2.4　十进制、二进制和十六进制

十进制并无特别之处，而计算机则采用二进制表示数字。这里，十进制并非是最佳选择。当然，十进制依然有其用武之地，例如，人类可利用 10 个手指计数。另外，由于 10 表示为 2 和 5 的倍数，因而可方便地确定某一十进制数字可被 2 或 5 整除。当然，通常难以判断此类数字是否可被 2 或 7 整除。然而，针对 3 或 9，则存在一类简单的方案可判断某一数值是否可被 3 或 9 整除（类似方案也存在于 2 或 5 中）。

【提示】当且仅当十进制各位数字之和可被 3 整除时，对应数值可被 3 整除。

相应地，其他进制情况又当如何？例如，十二进制广泛应用于货币测算以及英尺和英寸的换算中，并可方便地判断某一数字是否可被 2、3、4 或 6 整除。而六十进制常用于计时中，三百六十进制则用于测量角度。不难发现，六十进制和三百六十进制需要使用到更多的计数符号。巴比伦人曾采用了此类进制，尽管可通过十进制将数值进行细分，但其复杂度不言而喻。虽然颇具逻辑性，但十二进制的应用范围十分有限。

二进制通过多种方式体现了自身的有效性和优势。在二进制中，数字通过 2 的指数表示。当采用前述内容提及的进制位时，数字 11 可表示为 1011 或 $1\times8+0\times4+1\times2+1\times1$。正如十进制那样，当与整数协同工作时，可唯一地表示二进制数字。二进制的另一个特征是，当使用多位表达某一既定数字时，只需要两个符号即可。例如，1 表示为 1，3 表示为 11，5 表示为 101。在计算机体系结构课程中曾谈到，该数字关联方式可将 1 视为“开”，并将 0 视为“关”，即数字电路中基本的二进制语言。

二进制需要两个符号表达数字，除此之外，计算机操作中还包含十六进制。十六进制包含十进制 0～9 以及字母 A～F，后者表示为 10～15。据此，F1A 表示为 $15\times256+1\times16+10\times1$ 或 3866。二进制与十六进制之间的转换操作较为简单，各十六进制符号可直接转换为 4 个二进制符号。例如，F1A 可转换为 1111 0001 1010。

1.3　数字在计算机中的表达方式

二进制可视为计算机中数字表达的核心方法，本书也将围绕程序设计的数学理念展开讨论，因而深入分析计算机于二进制数字的使用方式将变得十分重要。

1.3.1　表达整数

由于二进制数字可通过两个符号表示，即 1 和 0 或“开”和“关”，因而针对计算机设备而言较为理想。再次强调，计算机仅考察数字数据，且此类数据以“开”、“关”集表示，并分别定义了信息的一位数据（此处，“位”即为二进制数位的简称）。相应地，8 个开关（位）可表达 0～255 范围内的数据；32 位开关则可表示 4294967295 之内的任意数字；而 64 位开关则可表示 18446744073709551615 范围内的数字，即四倍长字。

计算机本地环境的设计目标旨在通过二进制数字执行快速计算，且与处理器的位数相关。与 32 位机器相比，64 位设备则可处理更大的数字计算。需要说明的是，并非全部数据位均投入计算中。通常情况下，可设定某一位数据以表明当前数字为正数或负数。最终，32 位机器可处理 −2147483647～2147483647 范围内的数字，也就是说，最大正数范围并非是 4294967295。

鉴于加法和乘法运算的简单性，二进制具有较快的计算速度。下列伪代码显示了两个二进制字符串之间的加法运算：

```
function addBinaryStrings(b1, b2)
//pad out the smaller number with leading
//zeroes so they are the same length
set Output to ""
set CarryDigit to 0
repeat with Index = the length of b1 down to 1
set k1 and k2 to the Index'th characters of b1 and b2
if k1 = k2 then
//the digits are the same, so write the current carry digit
set WriteDigit to CarryDigit
set CarryDigit to k1
//if k1 and k2 are 1 then you are going to be carrying a digit
//if they are 0 then you won't.
otherwise
//the digits are different,
//so write the opposite of the current carry digit
set WriteDigit to NOT(CarryDigit)
//leave the carry digit alone as it will be unchanged
end if
append WriteDigit to the front of Output
end repeat
if CarryDigit = 1 then append 1 to the front of Output
return output
end function
```

上述算法体现了与计算机工作方式相关的核心内容，值得深入思考。在二进制中，01 + 01 = 10，01 + 10 = 11。由于根据各数据位相同与否执行加法运算（0 或 1），因而计算速度得到了较大的改观。此处，读者不必担心较大范围内的数据计算结果。鉴于得到了硬件的支持，计算机内部的整数运算相对简单。上述处理过程中的基础内容则是逻辑操作符 AND、OR 和 NOT，并采

用不同方式整合二进制（或布尔）数位。

若 A 和 B 表示为二进制数位，则 AND、OR 和 NOT 将以 3 种计算机逻辑控制方式转换最终结果，如图 1.1 所示。对于 A AND B，当且仅当 A 和 B 皆为 1 时，最终结果为 1；对于 A OR B，若 A 或 B 为 1，抑或二者皆为 1，则数据位最终结果为 1；对于 NOT 操作，仅当数据位设置为 0 时，数据位结果为 1。

AND

A	B	Result
1	0	0
0	1	0
1	1	1
0	0	0

OR

A	B	Result
0	1	1
1	0	1
1	1	1
0	0	0

NOT

Bit	Result
0	1
1	0

图 1.1　数据位中的基本布尔运算

二进制的乘法运算相对直观，该运算结合了两种操作，即基于 2 的乘法运算和加法运算。类似于十进制中的与 10 相乘的乘法运算，二进制中可通过移动一位实现基于 2 的乘法运算。例如，考察二进制中的数字 6，即 110，若该数字左移一位，对应结果为 12，其二进制表达形式为 1100。

1.3.2　有理数和无理数的表达方法

截止到目前，相关内容多集中于整数，但进制方案还可表示非整数数据。对此，可分别考察小于 1 和大于 1 的数据，对此，可使用小数点作为标记。其中，小数点构成了计数的起始点。当工作于十进制环境下，该小数点称作十进制小数点。其中，小数点后第 1 位称作十分位，小数点后第 2 位称作百分位，依此类推。在二进制中，二进制小数点后第 1 位称作 1/2 位，二进制小数点后第 2 位称作 1/4 位，依此类推。根据这一方案，数字 $1\frac{3}{8}$ 在二进制中表示为 1.011 (1 + 1/4 + 1/8)。

然而，该方案并非完美，且某些分数无法通过此类方法予以表达。例如，分数 1/3 无法通过十分位、百分位等数据位之和的方式表达，且该分数呈现为无穷级数之势，即 $\frac{3}{10}+\frac{3}{100}+\frac{3}{1000}\cdots$ 相应地，在十进制中，1/3 常表示为 0.333…

【提示】在数学领域内，术语“级数的极限”可实现准确的描述且具有重要的意义。在数字运算环境中，随着级数项的增加，最终累计结果将越发接近 1/3，且不会超过该值。

在二进制中，上述行为则变得更加糟糕。其中，分数通常难以表达，分母也仅限于 2。例如，在十进制中，1/5 可准确地记为 0.2；而在二进制中，该值则表示为无限重复型数字，即 0.001100110011…由于此类数字缺乏应有的截止点，因而计算机往往无法对其实现有效的处理。

有两种方式可避开这一问题。方式一是忽略小数点，并根据构成的整数数据表示分数，即分母和分子。随后，各有理数可实现准确的表达。尽管如此，该方案并非完美无缺。例如，若在 1/2 和 1/3 之间执行加法运算，最终答案为 5/6，且有 5/6+1/7 = 41/42，41/42 + 1/11 = 493/462，此类分数称作不兼容（incompatible）分数，且分母不包含公因子（第 2 章将对此加以讨论），因此当与此类分数协同工作时，需要增加分母的复杂度。最终，对应数字可超出计算机所定义的最大整数值。即使未到达最大值，随着数字的增大，计算机的计算速度也会显著降低。

该方案的另外一个问题则是无理数，且无法作为分数予以准确表达。对此，需要执行一类近似计算方案。为了获取较好的近似结果，计算机采用进制系统表达非整型数字，并将其舍入至最近值。

当然，问题远不止于此，下面考察定点数的表达方式。当采用此方案时，小数点后使用固定的数字集（例如 8），因而将通过有限数量的数字表达各数值，并可限制数值的大小。若小数点可变化，则对应数据称作浮点数，这也是大多数计算机语言采用的方案，且与科学计数法类似。

在科学计数法中，数字将使用首个（非 0）有效位表达数字，并于随后加入若干数字。这将明确指定小数点的位置，例如，201.49 可表示为(20149; 3)。实际上，该数字表示为 2.0149 e2。根据惯例，可在首个数字之后定位小数点并于此处计数。另外，读者还可前后移动小数点，例如，2.0149 e–3 可表示为 0.0020149，其中，e3 称作指数形式。

1.3.3　标准数字和计算数字

浮点数与科学计数法十分类似，且存在多种标准格式。对此，电气和电子工程师协会曾发布了一种格式，该组织专注于制定各种标准格式，并广泛地应用于计算机制造商中，此类标准对于不同硬件之间的公共语言十分重要。

当采用 IEEE 标准并与 32 位数据工作时，可采用 1 位表示符号（正数或负数），8 位表示小数点的位置，即−127～+127，其全部值为 255。实际数字略小于 2^{128}。其余 23 位则定义为尾数，或表示为数据的实际数位。由于二进制数字的首个有效位为 1，因此无须在计算机数字表达方式中将其包含在内，该位隐式存在，并提供了一个精度位。除了尾数之外，有效数也可用于描述浮点表达法中的实际数字。

【提示】需要注意的是，应区分指数以及精确度所体现的、不同数字的尺寸问题，这可通过有效数中的位数加以确定。虽然读者可表示较大数值，但其精确度未必会超过较小数值。例如，当采用 32 位浮点数时，通常无法区分 987874651253 和 987874651254。

为了进一步了解计算机对数字的处理方式，下面分别考察几个典型数值：

1. 数值 10.5

处理数值 10.5 包含如下步骤：

- 当表达数字 0.5 时，需要将其转换为二进制形式，即 1010.1。
- 在科学计数法中，0.5 表示为 1.0101 e3。
- 将数字 3 转换为二进制数字并加上 127（该偏移值可表示位于–127～127 范围内的数字，

而非仅是 0～255），进而得到指数数位 10000010。

- 利用 0 值填充 1.0101 并获取 24 位尾数 101010000000000000000000。
- 忽略固定的首位 1，因而 23 位有效数表示为 01010000000000000000000。
- 将 0 添加至最后一位，以使当前数据为正数。

2. 数值–1e35

处理数值–1e35 包含如下步骤：

- 通过小数形式表示–1e35。
- 在二进制科学计数法中，该值表示为–1.1010100101011010010110 1…e–117。
- 将–117 转换为二进制并加上 127，进而获得指数 1010，填充 0 值以得到 8 位数据 00001010。
- 有效数为小数点后的前 23 位，即 10101001010110100101101。
- 最后一位为 1，以表示当前数据为负数。

3. 结果分析

在上述操作过程中，唯一的问题是，忽略尾数的首位 1 将无法准确地表示浮点值 0。对此，当数字的指数部分为 0 时，则不再假设数据的首位为 1。这也意味着，若误差位于可接受范围内，此类数字可表示小于 2^{-127} 的数字。当采用这一非规范化方式时，该方案可表示低至 2^{-149} 的数字。

不难发现，可通过 0 有效数、0 指数定义的浮点数表达 0，正/负位则无硬性要求。除此之外，还存在某些特殊情况，例如上溢、下溢以及 NaN 值（Not a Number，非数值）。当数据包含无效结果时，通常会返回 NaN 值，例如–1 的平方根。

下面考察数字 2e–40 以说明上述数据的主要特征，相关步骤如下所示：

- 在二进制科学计数法中，2e–40 表示为 1.0001011011000010011000 1e–132。
- 由于–132 小于–127，因此无法通过常规精确度表达该数字。若将指数左移 5 位，则可得到 0.000010001011011000010011000 1e–127。
- 对此，可使用 00000000 的指数部分以及 00000100010110110000100 的有效数。尽管减少了 5 位，但却好过无法表示该数字。

在练习 1.2 中，读者需要编写一组函数并与 32 位浮点数协同工作。当前，计算机多采用 64 位格式表示数据，针对 32 位处理器，某些语言采用双精度数据，并将两个 32 位数字合而为一，进而生成 64 位数字，因而可描述更大范围的数据，并提高了数值的精确度。

1.3.4　公共函数

本节讨论大多数计算机语言涉及的常见函数，其中包括 abs()、floor()、ceil()以及 round()。某些语言采用了不同的名称定义此类函数，但其功能并无差异。此类函数较为常见，其重要性不言而喻，下面讨论其实现方式。

1. 绝对值

函数 abs()返回数字（浮点数或整数）的绝对值，其实现过程如下所示：

```
function abs(n)
  if n>=0 then return n
  otherwise return -n
end function
```

数字的绝对值表示为正值。具体而言，在 abs()函数中，若 n 表示为负值，则−n 为正值。当然，计算机的处理过程则较为简洁，也就是说，将浮点数或整数的符号位设置为 0。

2. 浮点数转换为整数

存在 3 个函数可将浮点数转换为整数，其特定功能如下所示：

- 对于数字 n，floor()函数计算小于 n 的最大整数。
- 对于数字 n，ceil()函数计算大于 n 的最小整数。
- 对于数字 n，round()函数计算最接近于 n 的整数。

图 1.2 显示了上述 3 个函数的计算行为，对应数值范围分别采用实线和虚线表示。例如，若 n 大于等于 3 且小于 4，则 floor(n)=3。

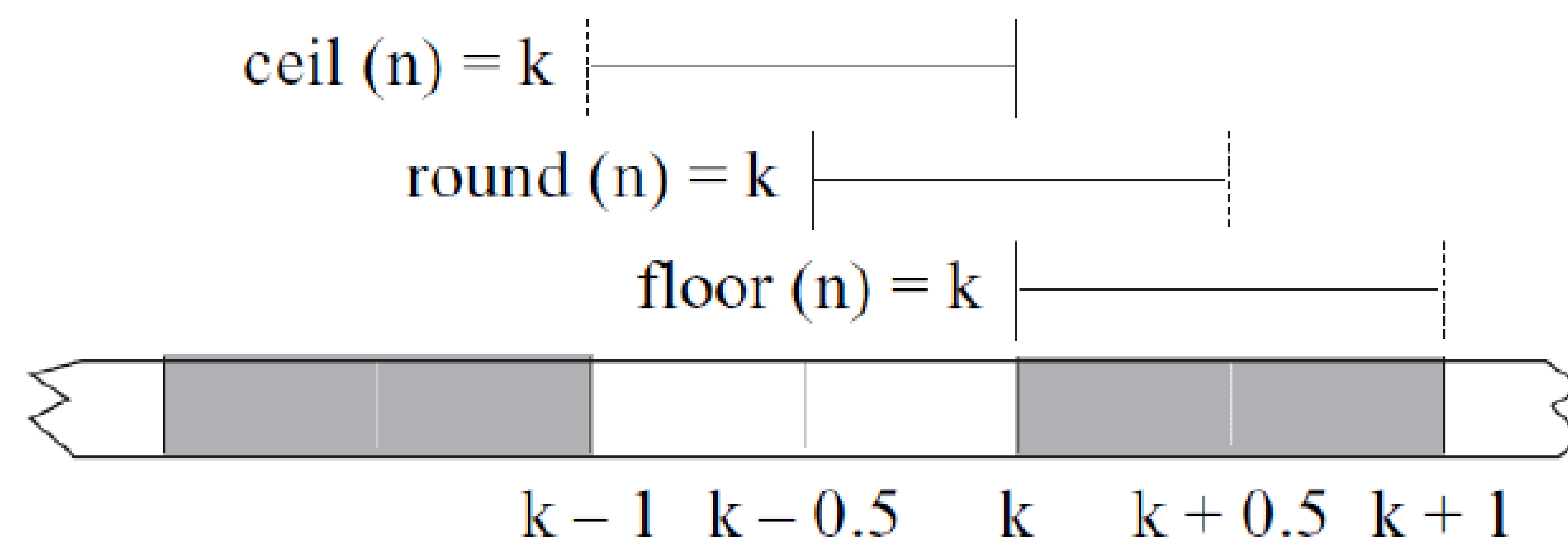


图 1.2　基于 floor()、ceil()和 round()函数的浮点数范围

其中，各函数应用于不同的场合。例如，当采用浮点表达方式时，对于计算机而言，floor()和 ceil()函数则易于计算。若对应数字的指数为 5，则函数可获取该数字的正整数部分（称作 p），即获取有效数的前 5 位；若数字小于 1，整数部分为 0；若数字大于有效数中的数位量，则函数无法得到准确的整数备份。随后，全部数字还需进一步检测符号值。若数字为正值，则 floor(n)等于 p，ceil(n)等于 p+1；若数字为负值，则 ceil(n)为$-p$，floor(n)为$-(p+1)$。

round()函数将数字舍入至最近值，并采用了大多数用户所熟悉的计算方式。该函数计算整数部分，且有 abs(n−round(n)) < 1。而位于两个整数之间的中值则不具有确定的含义。此时，round()函数定义为向上舍入，具体实现方式如下所示：

```
function round(n)
  Set f to floor(n)
  Set c to ceil(n)
  If n - f > c - n then return c // n is nearer c than f
  If c - n > n - f then return f // n is nearer f than c
  Otherwise return c // n is half-way between c and f
End function
```

round()函数展示了数字的舍入方式，并可方便地根据浮点表达形式进行计算。据此，读者可获取数字的整数部分 p。当确定向上舍入抑或是向下舍入时，还需考察有效数的下一个数位，即二进制中的 1/2。若该数位为 0，则数字的小数部分小于 1/2，这意味着向下舍入。若该数位为 1，

则小数部分大于或等于 1/2，进而可执行向上舍入。对于中值而言，向上舍入方案则更加方便。若采用向下舍入，则小数部分的首位为 1，因此若不考察其他数位，仍然无法知晓具体舍入方式。

1.3.5　舍入误差和性能

除了某些少数特定环境（例如 2 的指数形式），浮点数多少会存在误差。多数时候，这并不会产生重大问题，下面将对问题的具体情况加以分析。

例如，考察计算机计算$(\frac{7}{50}+1)\ 50-57$这一情形。当采用人工计算时，对应结果为 0，其中，$\frac{7}{50}\times50=7$，$1\times50=50$。然而，计算机返回的结果则是 7.10542735760100 e–15。其原因在于，当除以 50 时，数字包含负指数，加 1 后数字值包含 0 指数。这也意味着，有效数尾端多位将被丢弃，随后的乘法运算（乘以 50）并不能对其加以恢复。

当执行复杂计算和重复计算时，舍入误差可视为问题的主要来源。特别地，“拉伸以及折叠”计算将涉及小数与大数之间的加法运算。总体而言，此类问题并不需要用户过多干涉，但某些时候，用户应采用预计算方式以缓解此类问题产生的影响。

针对数据位的丢弃/恢复操作，为了避免此类问题，读者可尝试采用不同的顺序执行计算。在前述示例中，可考察$(\frac{7}{50}\times50)+(1\times50)-57$的计算结果。其中，除以 50 并于随后执行乘法运算将产生较小的精度损失，而基于较小数值（例如 50）的加法和减法运算则不会对运算产生负面影响。

另外一个需要注意的问题则是测试两个浮点数是否相等，例如$(\frac{7}{50}+1)\times5000=5700$。尽管二者应相等，但实际情况则与设想不符。通常情况下，仅当两个浮点数之差的绝对值小于某一较小值时，二者可视为相等。也就是说，应采用 if abs(x−y) < 0.00001，而非 if x = y。

尽管如此，上述措施依然无法完全消除此种误差。对于某些高精度计算，需要使用特定的算法，进而获取足够的精度，稍后将对此加以讨论。

与整数相比，浮点数计算相对耗时。例如，当采用浮点数计算 2×2 一百万次时，处理器的计算时间为 1.2 秒，而整数计算则低于 1 秒。对于此类简单任务，其差别十分明显。一种理论可描述为，对于游戏、密集型计算以及速度占优的操作，建议使用整数计算。

整数同样适用于分数计算。对此，可于先期乘以一个较大数，并转换为整数，在完成相关计算后，可再次除以该数值。这里，应谨慎处理此类计算，且期间应对计算结果予以测试。

当前，浮点数和整数计算之间的差异并不明显，且处理器可在硬件级别上处理浮点数，这一点与整数十分类似。鉴于二者间的差异日趋势微，读者应将注意力集中于其他代码的优化操作上。

1.3.6　BigInteger 类

对于高精度计算，一种处理方法是使用 BigInteger 类，该类生成特定对象，并将数值存储于

长数组中，而非计算机的处理器。前述内容曾使用长数位串表示二进制数字，该方案与 BigInteger 类较为类似。

当采用 BigInteger 类进行计算时，用户不再受限于处理器的内建数字存储机制。若计算机设备的内存空间允许，数组的长度则无硬性规定。最终，用户不再局限于 32 或 64 位数字表示方案，因此计算精度可得到巨大的提升。当然，鉴于计算源自程序涉及语言级别，而非处理器中，且各次计算涉及大量的数据位，该方案的计算速度较慢。在处理高精度计算时（例如头发、水流以及布料的物理模拟），计算机往往需要花费大量的时间对此进行处理。

大多数语言均提供了 BigInteger 类，例如 Java、C#以及 C/C++语言。其中，各类语言的处理手法类并无新奇之处，因此其差别并不明显。另外，本书涵盖的数学知识并不涉及 BigInteger 类。

1.4 本章练习

【练习 1.1】 试编写 convertBase (NumberString, Base1, Base2)函数，该函数接收一个字符串（或数组）umberString，即 Base1 进制中的某一整数，并将其转换为 Base2 进制，最终返回新的字符串（或数组）。这里，读者可尝试处理二进制至十六进制。

相应地，读者可对某些特例加以测试，例如二进制或十六进制，进而提升函数的计算速度。

【练习 1.2】 试采用 IEEE 浮点数编写函数，并实现加法、减法、乘法以及除法运算。其中，可将浮点数表示为 32 位的 1，0 字符串（数组）。

各函数应接收两个浮点数，并将计算结果返回至其中的一个参数中。此处应注意 0 指数这一特例。在实际操作过程中，读者将会发现，除法运算通常难于处理。

【提示】本书附录 E 提供了练习的提示内容。另外，本书合作网站也包含了部分练习答案以及伪代码。

1.5 本章小结

本章讨论了大量的基础内容，其中包括数字的理论处理方案、不同计算环境中计算机的解决方案、分数计算的精度问题，以及与整数相比，浮点数运算的耗时特征。在第 2 章中，本章理论将扩展至基础的运算操作中。

至此，读者应掌握如下内容：

- 整数、分数、有理数、分子、分母、定点数、浮点数、指数、尾数有效数、进制、二进制、十进制以及小数点等术语。
- 数值及其数位字符串表达方式之间的差异性。
- 浮点数和整数之间的转换方式。
- 浮点数的舍入误差及其消除方法。

第 2 章　数 学 运 算

本章包含如下内容：

- 概述。
- 分数。
- 比例、比率和百分比。
- 指数。
- 对数。

2.1　概　　述

本章和第 3 章将讨论基本的数学运算，以供后续学习使用。实际上，当读者为某一复杂问题一筹莫展时，其原因往往是未深入理解基本概念，理解事物的工作方式将为以后的学习打下坚实的基础。

本章讨论数学运算的必备知识，并与大多数计算机语言中的数据类型协同构造，即整数和浮点数。尽管此类数据类型并未包含全部数学运算，但却展现了计算中的常见问题。

本书合作网站提供了与本章内容相关的伪代码示例。

2.2　分　　数

第 1 章讨论了与分数相关的部分内容，本章将对此加以深入分析。下面首先讨论基本的数学运算，并围绕这一话题逐步展开深入讨论。

分数呈现为某种“指令”状态，并告知读者使用 a 除以 b。分数可通过多种方式予以实现，例如，分数 $\frac{2}{5}$ 等同于 $2 \div 5$ 这一形式。实际上，除号即形象地表达了当前的处理操作，其中，上下两点分别表示分数的分子（a）和分母（b）。

【**提示**】本书采用两种方式表达分数，方式一为 1/2，方式二则是 $\frac{1}{2}$。在书写时，后者可节省空间。

1. 分数间的乘法运算

分数间的乘法运算可通过分子和分母的乘法运算加以实现，如下所示：

$$\frac{2}{5}\times\frac{3}{7}=\frac{2\times 3}{5\times 7}=\frac{6}{35}$$

该运算包含两个步骤，即基于分母的除法运算以及分子间的乘法运算。针对分数与整数之间的乘法运算，由于整数可视为分母为 1 的分数，因而可将分数的分子与整数相乘，而分母则保持不变（乘以 1）。

2. 分数之间的除法运算

当使用$\frac{2}{5}$除以$\frac{3}{7}$时，可首先翻转$\frac{3}{7}$并得到$\frac{7}{3}$，即倒数，并于随后执行乘法运算，如下所示：

$$\frac{2}{5}\div\frac{3}{7}=\frac{2}{5}\times\frac{7}{3}=\frac{14}{15}$$

如何解释上述计算过程？对于初学者而言，可考察整数的操作行为。当某一数字除以 2 时，将二等分该数字（即获得该数字的半值）。不难发现，其行为等同于该数字与$\frac{1}{2}$之间的乘法运算。

在英语中，除法可采用多种方式描述，例如“per”——每小时英里数表示为行驶的英里数除以小时数。除此之外，除法还可通过“each”或“a”描述。

3. 分数间的加法和减法运算

与前述内容相比，分数的加法和减法运算则稍显复杂。当然，若两个分数的分母相同，则二者的加、减法操作十分简单，此时，分母保持不变，只须执行分子的运算即可。例如，对于加法运算，则有$\frac{3}{5}+\frac{1}{5}=\frac{4}{5}$；对于减法运算，则有$\frac{3}{5}-\frac{1}{5}=\frac{2}{5}$。其中，二者具有相同的单位，即$\frac{1}{5}$。若分数间包含不同的分母，则执行加法运算的对应单位也有所不同。

然而，分数可通过多种方式表达。例如，若分子和分母乘以某一数字，则可得到相同的分数，这是因为任何数字除以其自身值，最终结果均为 1。例如，分数$\frac{2}{2}$和$\frac{3}{3}$均为 1。最终，分数可乘以相关数值，且对应值保持不变。因此，通过选取便捷的表达方式，计算难度也将随之降低。

例如，假设执行$\frac{1}{2}+\frac{1}{4}$加法运算。其中，$\frac{1}{2}=\frac{2}{4}$（分子和分母均乘以 2），若采用此方式，则加法运算将变得十分简单。由于分数间包含相同的分母，因而分数加法运算演变为分子的加法运算，即$\frac{1}{4}+\frac{2}{4}=\frac{3}{4}$。该方法同样适用于转换过程并不明显的分数加法运算，例如$\frac{2}{3}+\frac{3}{5}$。

对此，可执行“交叉乘法”操作，即分数的分子和分母乘以另一个分数的分母。在计算语言中，该过程称作获取公分母，如图 2.1 所示。

$$\frac{2}{3}+\frac{3}{5}=\frac{2\times 5}{3\times 5}+\frac{3\times 3}{5\times 3}=\frac{2\times 5+3\times 3}{3\times 5}=\frac{19}{15}$$

图 2.1　交叉乘法

下面介绍上述操作在代码中的实现方式。为了便于理解，此处仅定义了伪代码函数，且并非是最终代码。该函数的名称为 addFractions()，并包含了两个参数，各分数通过二元素数组加以定义，如下所示：

```
function addFractions(f1,f2)
  set num1 to the numerator of f1
  set den1 to the denominator of f1
  set num2 to the numerator of f2
  set den2 to the denominator of f2
  if den1 = den2 then return the fraction(num1+num2, den1)
  otherwise
    set num3 to num1*den2
    set num4 to num2*den1
    return the fraction(num3+num4,den1*den2)
  end if
end function
```

该函数的计算过程稍显复杂。考察前述表达式$\frac{1}{2}+\frac{1}{4}$，addFractions()函数将计算$\frac{4}{8}+\frac{2}{8}$，而非更为简单的$\frac{2}{4}+\frac{1}{4}$。除此之外，函数的返回结果是$\frac{6}{8}$，而非$\frac{3}{4}$。从技术角度来看，该结果正确，但这并非是所需答案。

4. 因子和因式分解

在前述示例中，在执行加法运算之前，分数演变为更为复杂的形式。对此，一种较好的方法是在最后阶段将分数转换为简约形式，也就是说，将分数减至最低项，因而有必要对因子加以考察。在除法运算中，整数因子可得到整数结果，例如，1，2，3，6 均为 6 的因子。若 n 表示为 m 的因子，则称 m 为 n 的倍数。

【提示】数字 6 称作完全数，并等于全部因子（除了该数字本身）之和。例如，1，2，3 为 6 的因子，且有 $1+2+3=6$。值得一提的是，毕达哥拉斯对于完全数则是情有独钟。

对于分数$\frac{6}{8}$，将分子和分母同时除以 2，可得到$\frac{3}{4}$，这使得计算过程趋于简化。这里，数字 2 为分子和分母的因子，即 6 和 8 的公因子。实际上，6 和 8 不包含大于 2 的公因子，因此 2 为最大公因子，或者称之为最大公约数以及 GCD。相应地，分子和分母除以 GCD，即可将分数降至最小项。

【提示】某些时候。数字 n 和 m 的 GCD 可记为(n,m)。为了避免混淆，本书则采用 gcd(n,m)这一标识。

早期，GCD 出现于多个数字的质因数计算中，为了更好地解释质因数，下面首先考察质数。质数的因子只包含其自身和 1。根据惯例，质数列表中通常排除 1，其原因在于，1 往往以特例身份出现。同时，为了降低冗余性，质数通常包含 2，3，5，7，11，13，17，19，…。数学家们对此进行了大量的研究，旨在寻找某种分布模式以及指数的测试方式，这在密码学中十分重要。

5. 生成质数

读者可通过 Eratosthenes 筛选法生成小于 M 的质数列表。对此，可首先考察包含 2～M 的数值列表。这里，可选取表中的首个数字，并从当前列表中移除可被该数字整除的其他数字，该过

程重复执行，直至列表中的首个数字大于 $\sqrt{M}$ 。至此，列表中的剩余数字皆为质数。此类数字不可被小于平方根的任意数字整除，且大于 $\sqrt{M}$ 的两个数字的乘积，其结果必定大于 $\sqrt{M}$ 。

该算法的伪代码如下所示：

```
function listOfPrimes(M)
  //make an array with all the numbers from 2 to M and call it nlist
  set maxM to floor(sqrt(M)) //the biggest number that must be checked
  set index to 1
  repeat while nlist[index] is less than maxM
    set prime to nlist[index]
    repeat for index2 = index+1 to the number of elements in nlist
      if nlist[index2] is divisible by prime then delete it from the list
    end repeat
    add 1 to index
  end repeat
  return nlist
end function
```

不难发现，数字 n 的质因数为质数，且该质数为 n 的因子。这里，各数字均可通过质数积这一唯一方式表达，例如 $12 = 2 \times 2 \times 3$。这一事实体现了算术基本定理中的重要内容。另外，包含多个质因子的数字则称作合数。

【提示】在算术基本定理中，数学家们将 1 排除在质数之外。否则，这将视为唯一性规则的一个特例。读者可将任意数字表示为其质因数与多个 1 之间的乘积。

当计算任意数字 n 的质因数时，一种相对可靠的方法是将该数字持续除以质数，直至获得最终结果。当然，对于较大的数字，该过程的计算速度较慢；而对于较小的数字，该方法工作良好，当质数表生成完毕后尤其如此。对应伪代码如下所示：

```
function primeFactors(n)
  // create a list of possible primes
  set plist to listOfPrimes(floor(n/2))
  set rlist to a blank array
  set index to 1
  repeat while index <= the number of elements in plist
    set prime to plist[index]
    if n is divisible by prime then
      set n = n/prime
      add prime to rlist
    otherwise
      add 1 to index
    end if
  end repeat
  return rlist
end function
```

当计算极大的质数时（例如加密等功能），由于耗时较长，因此上述方法难以奏效。相反，可通过相关技巧反复测试质数，并在一定程度上获取近似结果。该方法可满足大多数需求，当然，

实际情况还取决于具体的应用程序。

6. GCD 和欧几里德算法

针对两个数字的 GCD 问题，若计算 gcd(n, m)，则可计算各数字的质因数。这里，假设 n 为 24，m 为 60。通过上述函数，n 的质因数为 2，2，2，3；m 的质因数为 2，2，3，5。随后，可从两个质因数序列中获取最大数量的匹配元素，即两个 2 和一个 3。因此，gcd(n, m)表示为 $2\times2\times3=12$。

欧几里德算法相对简单且颇具技巧性，并可节省大量的计算时间。

欧几里德算法源自简单的观测过程。针对数字 a，b，c，有 $a=b+c$，若 d 表示为 a 和 b 的因子，则 d 也为 c 的因子。

据此，可设计另一个计算函数 gcd(n,m)，其中，$n\geqslant m$，并利用 n 除以 m 进而获得余数 r。针对某一 a 值，有 $n=a\times m+r$。其中，$r<m$（即余数定义）。若 $r=0$，则 m 表示为 n 的因子，则有 gcd(n, m) = m。否则，根据定义，GCD 为 n 和 m 的因子，因此必为 r 的因子。随后，可利用 m 和 r 替换 n 和 m，进而重复上述处理过程。该函数的伪代码如下所示：

```
function gcd(n, m)
  set r to the remainder of n/m
  if r = 0 then return m
  otherwise return gcd(m, r)
end function
```

最终，数字 d 表示为两个参数的约数。随后，经回溯计算后可知该值为原数字的公约数。在数学领域内，Euclid 算法可视为一类完美的解决方案。

7. 最小公倍数

待 gcd(n, m)计算完毕后，读者还可进一步计算 n 和 m 的最小公倍数（LCM），该值等于 n*m/gcd(n, m)。LCM 常出现于分数的求和计算中。

当两个分数执行加法运算且不采用交叉乘法方法时，一类高效的方案则是计算最小公分母。换言之，分数的分子和分母均乘以一个最小数字，以使分数具有相同的分母，这里，最小公分母即为两个分母的 LCM。

下面讨论 $\frac{7}{24}+\frac{7}{30}$ 的计算过程。此处并不打算使用交叉乘法方法，其中，分数乘以某一数字，该数字可描述为：另一个分数的分母除以两个分母的 GCD，具体过程如下所示：

- 根据 Euclid 算法，gcd(24, 30) = 6，因此第一个分数乘以 $\frac{30}{6}=5$，第二个分数乘以 $\frac{24}{6}=4$。
- 当前结果为 $\frac{35}{120}+\frac{28}{120}$，其中包含了公分母 120（即 24 和 30 的 LCM）。
- 当使用公分母时，经简单的加法运算后可得到 $\frac{63}{120}$。
- 当前，可通过计算 gcd(63,120)将分数化为最简值。根据 Euclid 算法，$120=63+57$，$63=57+6$，$57=9\times6+3$，$6=3\times2$，因此 gcd(120, 63) = 3。

- 上述分数的分母和分子均除以 2，进而得到$\frac{21}{40}$，即最简分数。

【提示】由于 gcd(21, 40) = 1，因此分数 21/40 称作最简分数。此时，21 和 40 彼此互质。

8. 求模计算

Euclid 算法在两个数字相除时将计算余数，该过程十分有用，且存在单独的术语对其加以描述。若 n 除以 m 后的余数为 r，则称 n 全等于 r mod m。其中，全等符号记为“≡”，则有 $n \equiv r (\text{mod } m)$。

实际上，全等关系并不止于此。例如，当使用 5 执行求模计算时，4，9，14…彼此相等，且全等于 4。

大多数程序设计语言通过模函数对此予以支持，此类函数接收两个参数 n 和 m，并返回余数 r，例如 mod (n,m) = r。

若 $n \geqslant 0$，则 mod()函数返回位于 0～$m-1$ 之间的数据；若 $n < 0$，该函数返回位于 $-(m-1)$～0 之间的数据（至少在大多数计算机语言中的确如此）。考虑到该计算的普遍性，下面介绍该函数的修正版本，其伪代码如下所示：

```
function StrictModulo(n, m)
  set r = mod (n,m)
  if r<0 then return r + m
  otherwise return r
end function
```

StrictModulo()函数返回 0～$m-1$ 之间的数据值，全等关系 $n1 \equiv n2 (\text{mod } m)$等价于下列代码：

```
StrictModulo(n1, m) = StrictModulo(n2, m)
```

尽管模函数用途广泛，但该函数的常见应用是判断两个数字之间是否可整除。若 n 可被 m 整除，则有 StrictModulo(n, m) = 0。

【提示】模函数并不与 0 对称。总体而言，StrictModulo(n, m)不等于 StrictModulo(−n,m)。实际上，除非 n 为 m 的倍数（此时二者皆为 0），通常情况下则有 StrictModulo(n,m)=m−StrictModulo(−n,m)。

除此之外，读者还可尝试使用另一种方案执行求模计算。如前所述，floor(n/m)可生成小于 n/m 的最大整数。随后，m*floor(n/m)表示为小于 n、m 的最大倍数。这意味着，对于某一正整数，有 mod(n,m) = n−m*floor(n/m)。此处应注意浮点数和整数之间的转换问题，因而不应将该方法视为通用规则。

9. 数据循环

由于模运算常用于小型选项表中的循环遍历操作，因而该运算有时也称作时钟运算。时钟表盘仅包含 12 个数字，因此当计算小时时，有 8+6=2，这也是模运算在生活中的实际应用。

在程序设计中，可通过模运算处理此类问题。例如，当创建时钟对象并对时间值执行累计计算时，若不采用模运算，则对应伪代码如下所示：

```
function naiveClockAdd(oldHours, oldMinutes, addHours, addMinutes)
  set newMinutes = oldMinutes + addMinutes
  repeat while newMinutes>60
    //drop the number of minutes by 60 and add another hour
    subtract 60 from newMinutes
    add 1 to addHours
  end repeat
  set newHours = oldHours + addHours
  repeat while newHours>12
    //drop 12s until you're in the range 1-12
    subtract 12 from newHours
  end repeat
  return array(newHours,newMinutes)
end function
```

尽管上述函数并不复杂，但使用模运算时，函数的整体风格则更加简约，如下所示：

```
function cleverClockAdd(oldHours, oldMinutes, addHours, addMinutes)
  set newMinutes = strictModulo (oldMinutes + addMinutes, 60)
  add (oldMinutes + addMinutes - newMinutes) / 60 to addHours
  set newHours = 1+ strictModulo (oldHours + addHours - 1, 12)
  return array(newHours, newMinutes)
end function
```

第 1 行代码使用了新定义的分钟值，该值位于 0～59 范围内，并通过(oldMinutes + addMinutes)予以计算，其结果针对 60 执行求模运算。另外，代码还将对小时数进行计算。回忆一下，当 n 除以 m 时，strictModulo(n,m)表示为余数结果，这也意味着，对于 a 值，有 n = a*m + strictModulo (n,m)。当计算 a 时，可从 n 中减去 strictModulo(n,m)并除以 m。由于 a 表示为小时数（即 60 分钟的倍数），故可通过该值递增 addHours 变量。

【提示】尽管该方法工作良好，但此处应采用 floor(n/m)函数计算 a。

cleverClockAdd()函数的第 3 行代码计算 newHours 变量值，唯一复杂之处在于该值位于 1～12 范围内，而非 0～11。对此，可在模运算之前从 newHours 中减去 1，最终结果值将小于 0～11 间的实际小时数。随后，可再次加上 1，以使最终答案位于所需范围内。

然而，上述计算过程稍显晦涩。多数时候，模函数根据下列 3 项内容产生变化：

- 通过 mod(n,m)将 n 减至 0～m–1 范围内（数据循环）。
- 从 n 中减去 mod(n,m)，以使 n 可被 m 整除。
- 在调用 mod(n,m)之前预计算 n（预减或预加），并于随后再次加/减该值，以使其位于期望范围内，特别是 1～m。

关于模函数，另一个示例则是大型数据集的初始化操作，例如网格的构造。这里，假设 $n \times m$ 对象数组表示宽为 n、高为 m 的网格，各对象具有数组索引，并计算其在网格中的位置，模函数可满足这一计算需求，对应伪代码如下所示：

```
function positionInGrid (squareNumber, numberOfColumns)
  set positionAcross to 1 + mod(squareNumber-1,numberOfColumns)
  set positionDown to (squareNumbermod-
    mod(squareNumber, numberOfColumns)) / numberOfColumns
  return array(positionAcross, PositionDown)
end function
```

2.3　比例、比率以及百分比

分数的另一种常见解释为分子和分母之间的比率，以及百分比，本节将详细介绍这两个概念。

2.3.1　数值范围间的映射

通常情况下，比率（或比例）与分数具有相同含义，当关联类似对象时尤其如此（“类似”的准确含义将在第 5 章介绍）。比率有时也采用冒号形式书写，例如 4:3，并与 4/3 等价。又如，如果一个对象的尺寸是另一个对象的 4/3 倍，则二者的尺寸比率为 4:3。通常情况下，比率用于描述对象的划分方式，例如，将蛋糕按照 1:2 切割，即一块蛋糕的尺寸 2 倍于另一块蛋糕。由于 $\frac{1}{1+2}+\frac{2}{1+2}=\frac{1+2}{1+2}=1$，因此分割后，一块蛋糕占据 1/3，另一块蛋糕占据 2/3。

2.3.2　纸张尺寸

比率在处理对象的缩放操作时十分有用，例如，若纸张的宽度为 297mm，高度为 210mm（即标准的 A4 纸），若尺寸加倍，则各条边均保持相同比例，其原因在于，若分数的分子和分母乘以同一数值，则分数大小保持不变。

当对物体执行缩放操作时，这一结果十分有用。如果高度与宽度的原始比率已知，则可根据既定宽度值计算高度值，全部工作只须实现新宽度与比率之间的乘法运算即可。对应伪代码如下所示：

```
function newHeight(originalWidth, originalHeight, newWidth)
  return newWidth * originalHeight / originalWidth
end function
```

下面从另外一个角度考察该问题。当缩放任意对象时，该对象的各个量度（包括对角线、高度、宽度）均乘以一个相同的量值。若原值或新值已知，则乘以比率即可得到其他尺寸。例如：

```
newDiagonalLength = originalDiagonalLength * newHeight / originalHeight
```

该结论可方便地生成两个矩形之间的映射函数。若已知矩形内的点坐标，通过与比例值相乘，

即可将其与另一矩形中的等价点进行关联。

纸张的尺寸常以 A 开头，当在水平方向上（竖排格式）均分纸页时，A 后的序号加 1（如图 2.2 所示），其代数表达式如下所示：

$$\frac{aHeight}{aWidth}=\frac{aWidth}{(aHeight/2)}$$

经整合后，可得到下列等式：

$$aHeight^2 = 2\times aWidth^2$$
$$\frac{aHeight}{aWidth}=\sqrt{2}$$

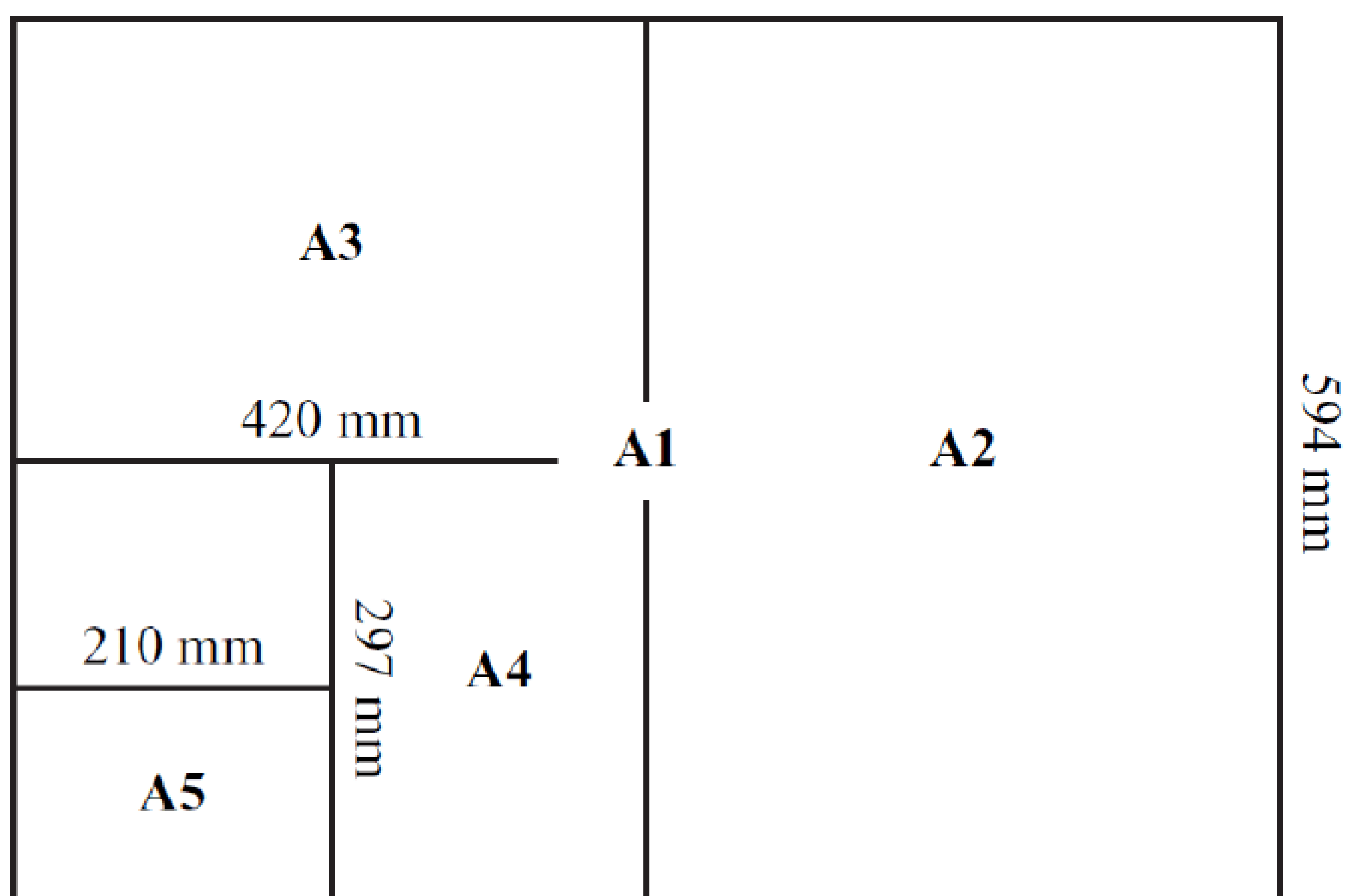


图 2.2　纸张尺寸

在纸张的各种尺寸中，高度值表示为 $\sqrt{2}$ =1.4.1…乘以宽度值（或近似值）：$\frac{297^2}{210}$=2.0002…图 2.2 显示了基于比率值的纸张缩小状态。

2.3.3　黄金比率

另一种有用的比例尺寸为黄金比率，并采用希腊字母 φ 表示。作为一种矩形比例，若从一侧切除正方形部分，则剩余部分与原始部分具有相同的比例。此处并不打算解释其中的代数内容，对应结果表示为 $\varphi=\frac{1+\sqrt{5}}{2}$=1.618…。在图 2.3 中，其比例模式与图 2.2 有几分类似。

古希腊人和其他思想家发现了与 φ 相关的诸多特征，并将其称作神圣分割比例或黄金分割等。其中，在画布上根据该比例放置的物体具有较好的美学特征，该结论在一定程度上正确（例如帕特农神庙），但最近的心理学测试表明，这一结论并不具备普遍意义。针对标准纸张尺寸，比例 1:sqrt(2)已经非常接近流行的尺寸。另外，当前人们的审美已逐渐接受了 A1、A2、A3、A4 这一类纸张的尺寸。

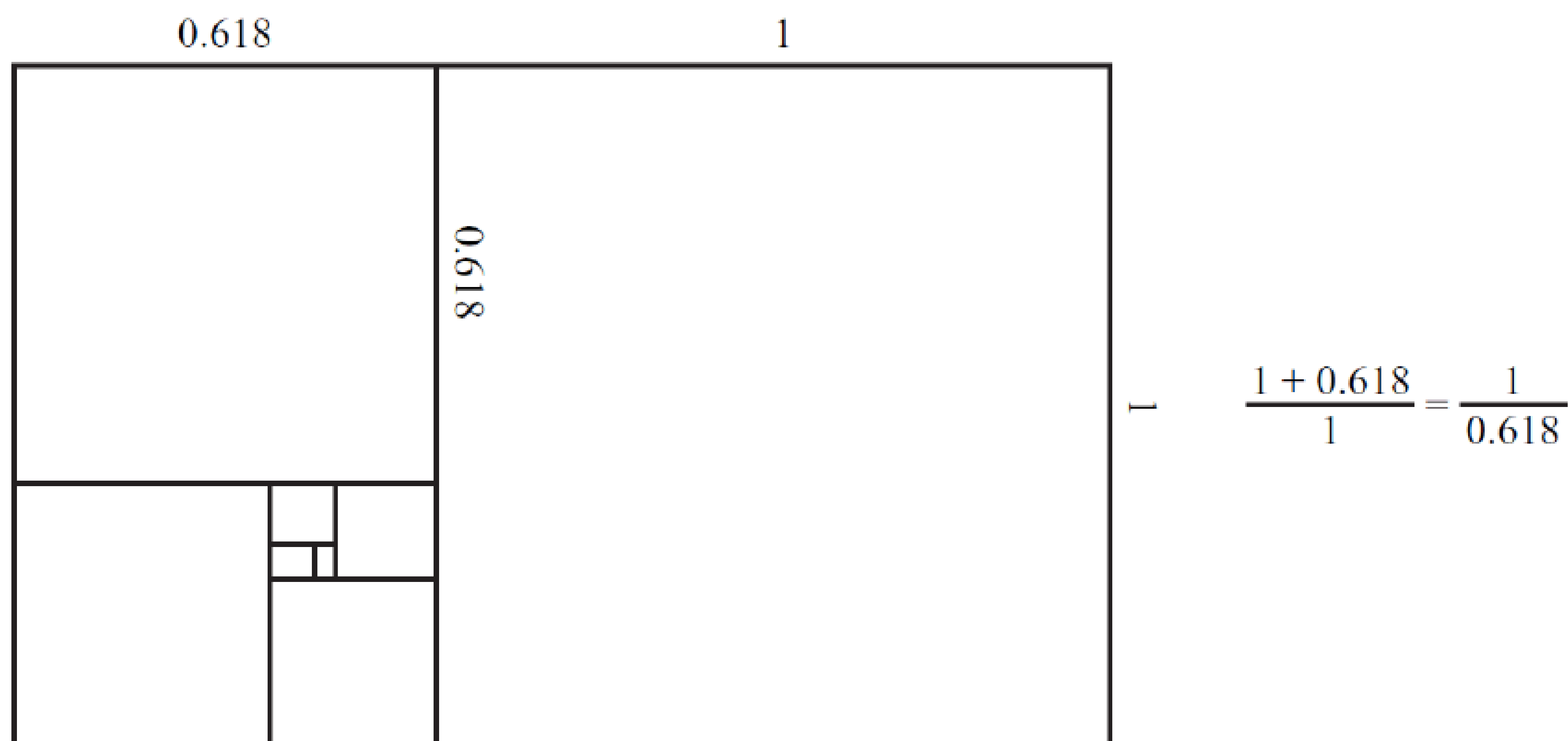


图 2.3　黄金比率

2.3.4　Fibonacci 数列

谈到黄金比率，则不可不提 Fibonacci 数列。Fibonacci 数列表示为由 1，1，2，3，5，8，13，21，…构成的数字序列，其中，各数字均为前两个数字之和。同时，连续项之间的比率将迅速接近于 φ 值。根据 Fibonacci 数列定义，任何数字序列均具有此类特征。在科学领域中，Fibonacci 序列证实了许多有趣的现象，例如，贝壳的凹线以及藤本植物中叶子的分布状态。

2.3.5　滑块

滑块可视为比例的又一个应用场合。这里，滑块表示为数字范围的图形表达方式。在标准的滑块中，直线表示数字的范围（例如 10～100），可移动的指针则体现了当前数据值（如图 2.4 所示）。

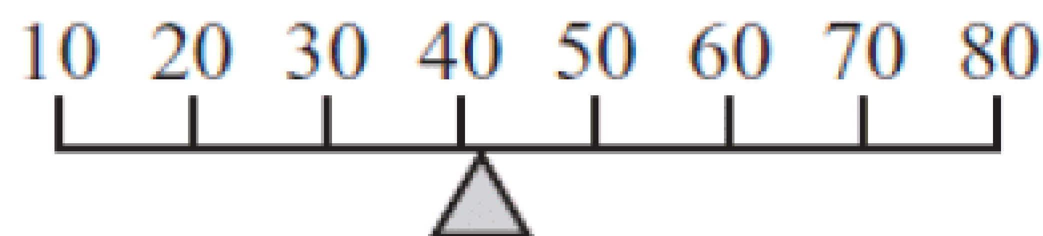


图 2.4　标准的滑块

当构造滑块时，须知晓 4 个数据值，即直线两端的位置及其所表示的最大值和最小值。当前，读者无须了解直线两端的“位置”的含义，并可将其视为一个数字，例如一端为 100，另一端为 200。

当对滑块执行初始化操作时，可快速计算其固有的比例——利用滑块的一个单位表示值范围，并采用全部范围值除以滑块长度计算该比例值，如下所示：

$$intrinsicProportion = \frac{maxValue - minValue}{endPoint2 - endPoint1}$$

当计算滑块上特定点所表示的数值时，可计算该值沿滑块的距离，将其乘以固有比例并与最小值相加，如下所示：

$$Value = (thisPoint - endPoint1) \times intrinsicProportion + minValue$$

需要注意的是，此处的计算方式与缩放后的纸张比例计算类似。从本质上讲，二者具有相同的计算过程。也就是说，当前操作生成了一个虚拟纸张页面，其宽度值为滑块的长度，高度为所表达的数据值范围。当计算新点时，可采用新的宽度值以及相同比例缩小至另一个矩形。当然，矩形无须从 0 开始，因而需要加上最小值。

若读者打算表示某一特定值，则可通过计算距最小值的距离获取在滑块上的位置，除以固有比例并加上端点，如下所示：

$$newPoint = \frac{thisPoint - minValue}{intrinsicProportion} + endPoint1$$

如果读者难以确定通过除法抑或是乘法完成计算，则可尝试使用某一端点以观察计算结果，例如，可将 $thisPoint$ 设置为第 1 个公式中的 $minValue$，进而得到 $newPoint = 0 + endPoint1$，即第 1 个端点，若将 $thisPoint$ 设置为 $maxValue$，则可得到如下等式：

$$newPoint = \frac{maxValue - minValue}{intrinsicProportion} + endPoint1$$

回忆一下，除以一个分数相当于乘以其倒数，因而上式转换为下列等式：

$$newPoint = (maxValue - minValue) \times \frac{endPoint2 - endPoint1}{maxValue - minValue} + endPoint1$$

在上述分数中，$(maxValue - minValue)$项被消除，对应结果如下所示：

$$newPoint = endPoint2 - endPoint1 + endPoint1 = endPoint2$$

上式表示为第二个端点。相应地，读者还可针对第二个公式执行相同的检测。

滚动条可视为一类特殊的滑块，并可在较大的图像区域中表示较小图像的位置，其原理与标准的滑块基本相同，但复杂之处在于，较小图像的尺寸将发生变化。例如，可尝试滚动一段文本内容，若已知文本的像素高度以及滚动窗口的高度，则滚动栏可视为数值位于 0（顶端位置）和 $textHeight \neg windowHeight$（底端位置）之间的滑块，这一类数值表示窗口上方文本的像素位置。当文本位于 $textHeight \neg windowHeight$ 位置时，则窗口底部（距 $windowHeight$ 个像素）位于 $textHeight$ 处，即文本底部。

若在标准窗口中考察滚动栏，则会发现滚动标记的尺寸也会发生变化，进而展示图像观察区域的大小，对应比例为 $windowHeight/textHeight$（参见练习 2.1）。

2.3.6　百分比计算

百分比可视为另一种分数，并可视为分数计算的直接扩展，例如 20%等于 $\frac{20}{100}$（百分比意即百分数或“百分之”）。具体而言，其计算形式如下所示：

- 1000 的 20%等于 20/100 乘以 1000，即 200。
- 1000 去除 20%等于 1000– 20%×1000，即 1000 – 200 = 800。
- 1000“超额”20%后的总结果为 1000+ 20%×1000，即 1000 + 200 = 1200。

其中，最后两项计算可分别简化为 1000 乘以 80%和 1000 乘以 120%。

例如，若商品从25美元降至20美元，则可计算其降低率。这里，商品降低了5美元，即原始价格的$\frac{5}{25}$，由于$\frac{5}{25}\times 100=20$，因而降低率为20%。

2.3.7 复利计算

若每年向账户中存储固定百分比的资金，则复利计算过程稍显复杂。例如，若客户存储1000美元，且利息为3%，则10年到期后账户余额是多少？或许有些读者认为每年累加1000×3%=30美元即可。这里的问题是，第2年的账户余额不再是1000美元，而是1030美元，因而该百分比计算无法得到正确的答案，进而需要计算增长量。

下列输出内容源自mortgates.html文件中的代码，并生成了一个10年期限的账单：

```
At the end of year 1 you have 103% of $1000 = $1030
At the end of year 2 you have 103% of $1030.00 = $1060.90
At the end of year 3 you have 103% of $1060.90 = $1092.73
At the end of year 4 you have 103% of $1092.73 = $1125.51
At the end of year 5 you have 103% of $1125.51 = $1159.27
At the end of year 6 you have 103% of $1159.27 = $1194.05
At the end of year 7 you have 103% of $1194.05 = $1229.87
At the end of year 8 you have 103% of $1229.87 = $1266.77
At the end of year 9 you have 103% of $1266.77 = $1304.77
At the end of year 10 you have 103% of $1304.77 = $1343.92
```

与每年累加30美元相比，此处应注意复利的增长速度。10年后，与每年累加30美元相比，实际余额将多出43.92美元。最终百分比增长为$\frac{43.92}{300}\times 100 = 14.64\%$，与前述简单的利息累计相比，客户的收入增长了近15%。

由于利息以指数方式增长（每年乘以同一量值），而非线性增长（每年累加同一量值），因而其变化速度较快。稍后将对指数进行深入分析，当前计算可总结为如下方式：n个月后，现金余额为$initialCash \times increase^n$，其中，$increase$表示为百分比增长，即$\frac{(100 + interest)}{100}$或$1 + \frac{interest}{100}$。

2.3.8 债务和利息

债务利息也存在类似情况。例如，抵押贷款的复杂之处在于，债务的偿还行为全程有效。如果读者使用过“抵押贷款计算器”，则需要输入抵押贷款量、利息率以及偿还年限，这可视为一类较为复杂的百分比计算。需要说明的是，此处讨论的内容可视为资金和偿还抵押贷款中的一个特例，最终结果将随国家和政策制定者的不同而变化。

考察下列示例，针对100000美元抵押贷款，若月首付1000美元，利息率为5%，则多久可偿还清抵押贷款？

首先，需要将利息率转化为月度形式。当前，年利息率为$\frac{5}{100}$或0.05，对此，可除以12得

到月利息率，即 0.00417。需要注意的是，由于当前操作为复利计算，因而实际年利率超过了 5%，即 1.0041712–1=0.0511 或 5.11%。

月底时，债务增长了 0.00417。换而言之，原额度乘以 1.00417，进而增至 100417 美元。随后，支付额度从债务中扣除，因而对其贷款额度为 99417 美元。在每一个月中，客户需要计算 *newLoan*=*oldLoan*×1.00417–1000。

当使用上述算式时，对应结果如下所示：

```
At the end of year 1 the loan is $92837.33
At the end of year 2 the loan is $85308.21
At the end of year 3 the loan is $77393.89
At the end of year 4 the loan is $69074.65
At the end of year 5 the loan is $60329.79
At the end of year 6 the loan is $51137.52
At the end of year 7 the loan is $41474.95
At the end of year 8 the loan is $31318.03
At the end of year 9 the loan is $20641.47
At the end of year 10 the loan is $9418.67
At the end of month 130 the loan is $0
```

通过观察可知，贷款于 10 年还清，全部偿还额度为 129628.96 美元，增长了约 30%。

上述计算还可通过下列公式实现，当然，其内容稍显复杂：

$$debtAfterNMonths = initialAmount \times I^n - monthlyPayment \times (I^{n-1} + I^{n-2} + \cdots + I + 1)$$

其中，I 表示为月度增长，如下所示：

$$1 + \frac{annualInterest}{1200}$$

另外，$I^{n-1} + I^{n-2} + \cdots + I + 1$ 序列可视为几何级数中的一个特例，其值为$\frac{I^n - 1}{I - 1}$，因而最终公式如下所示：

$$monthlyPayment = initialAmount \times \frac{I - 1}{I^n - 1}$$

其中，$n = numberOfYears \times 12$。

因此，若贷款 1000000 美元，为期 10 年且利率为 5%，则用户需要月付 1060.66 美元，这也是诸多在线计数器的计算结果。

2.4　指　　数

前述内容已多次使用指数函数，总体而言，该函数表示数值自乘 n 次。尽管这一描述尚可令人接受，但指数函数的计算方式依然值得深入探讨。

2.4.1　指数计算

如何理解数值自乘–0.3 次？该问题貌似毫无意义。然而，若输入 power(2,−0.3)，则计算机将

会输出 0.81。此处，负分数指数形式并未违背原始定义。为了进一步了解指数的计算方式，读者应考察下列基本问题。

（1）如何理解 n^{p+q}？

该表达式意味着 n 自乘$(p+q)$次，也就是说，首先，n 自乘 p 次；随后，n 自乘 q 次；最后，两个计算结果执行乘法运算，即 $n^{p+q}=n^p\times n^q$。

（2）如何理解 $n^{p\times q}$？

该表达式意味着 n 自乘 $p\times q$ 次，这相当于 n 自乘 p 次，对应结果再次自乘 q 次，即 $n^{p\times q}=(n^p)^q=(n^q)^p$。

（3）如何理解 n^0?

根据首个指数加法公式，针对任意 p，由 $n^{0+q}=n^0\times n^p$，但 $n^{0+p}=n^p$，因而针对任意 n，有 n^0=1。

（4）如何理解 $(n\times m)^p$？

该表达式意味着$(n\times m)$自乘 p 次。也就是说，n 自乘 p 次，m 自乘 p 次，而后二者再执行乘法运算，因而有 $(n\times m)^p=n^p\times m^p$。

（5）如何理解 $(nm)^p$？

可以确定的是，该表达式表示为 nm 自乘 p 次，即 $n^m\times n^m\times\cdots\times n^m$。根据首个指数加法公式，该式等于 $n^{m\times p}$。

（6）如何理解 n^{-p}？

根据首个指数加法公式，$n^{p-p}=n^p\times n^{-p}$。但 $n^{p-p}=n^0=1$，因而有 $n^{-p}=\dfrac{1}{n^p}$。

（7）如何理解 $n^{1/p}$?

首先考察$(n^{1/p})^p$，该式等于 n^1，即 n。换而言之，$n^{1/p}$ 的 p 次幂等于 n，因而 $n^{1/p}$ 为 n 的 p 次根。相应地，n 的平方根表示为 $n^{1/2}$。

综上所述，负指数或分数指数与整数指数的定义保持一致。实际上，读者可定义任意形式的指数函数，包括虚数、矩阵，甚至是函数自身。

需要注意的是，在实数范畴内，负数的分数指数往往会出现问题，例如，–1 不存在平方根。另外，立方根也存在类似的情况。总体而言，计算机或计算器通常会禁止用户计算–1 的平方根。若尝试计算 power(–1.1/3.0)，用户通常不会得到相应的答案——设备通常往往会禁止此类操作。鉴于计算机无法准确表达分数 1/3 的准确值，因而禁止上述操作也在情理之中。如果读者尝试计算负数的立方根（或其他根值），则可计算该数据绝对值的根值，并对结果进行检测，对应伪代码如下所示：

```
function mthRoot(n, m)
  if n<0 then
    set p to power(-n, 1.0/m)
    if abs(power(-p, m)-n)<0.1 then return -p
    otherwise return "No such value”
  otherwise
    return power(n,1.0/m)
  end if
end function
```

当处理舍入误差时，该函数采用了最近值检测，而非浮点数的等值操作。

2.4.2　数字 e 和 exp()函数

当执行指数计算时，很可能遇到 e 值，该数字约为 2.718。e 值包含较为特殊的属性，稍后将对其加以深入讨论，目前，读者仅将其视为一个数字即可。需要注意的是，数字 e 等于无限求和公式 $\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}\cdots$（这里，“!”表示为阶乘函数 $n! = 1\times2\times\cdots\times n$，且有 0!=1）。

从程序员视角来看，数字 e 多与 exp()函数有关，其中，exp(x)等于 e^x。为了得到数字 e 的值，可尝试输入 exp(1)，并可将其视为指数的“标准”。

【提示】读者不应将数字 e 与第 1 章中所讨论的指数符号混淆，例如 1.53e6——表示小数点移动的位数。从数学角度来看，这等于尾数与当前进制指数结果之间的乘积。若采用十进制，则有 1.53×10^6。

2.4.3　真实世界和物理学中的指数函数

假设玛丽在后院种植了一株藤本植物，目前该植物长 1m，其长度以每星期 20%的速度增长。若以同一速率增长，1 年以后，该藤本植物的长度是多少？

该问题类似于前述抵押贷款示例，具体结论可描述为：一段时间内，若某一值乘以持续增长率若干次，该值通常具有较快的增长速度，即指数增长，常见于人口、生物以及经济增长计算。

具体而言，1 个星期后，藤本植物的长度为 1.2m；两个星期后，其长度为 $1.2\times1.2 = 1.44$m；52 个星期后，长度值为 $1.2^{52} = 13104.63$m，这已然超过了 13km，这形象地解释了几何增长的含义。几何增长速度通常较快，即使较小的增长率也具有惊人的累计效果。例如，若增长率为每星期 5%，则 1 年后，藤本植物的长度将超过 12m。

相反，若数据以固定因子减小，则称作指数衰减。在放射学领域内，放射元素常具有半衰期，即原子核半数衰减所需的时间。与快速增长的指数增长不同，指数衰减通常较慢。实际上，由于该过程难以完成，因而可视为一个无限慢速的过程。例如，若某一数字乘以 1/2，则该数字将逐渐减小，但永远不会为 0。从数学角度上讲，该过程的极限值为 0。

人们往往对辐射衰减感到陌生，对于半衰期为几百年（或几千年）的放射元素，其衰减速度较慢，因而放射特征并不十分明显，此类元素还将历经较长的放射期。相反，某些元素则具有强烈的反射性特征以及较短的半衰期。相应地，此类元素仅在较短时期内具有危害性，经过快速衰减后，可转变为相对无害物质。其中，较具危险性的物质则是具有中间半衰期的一类物质，例如锶-90，其半衰期约为 30 年。

2.5　对　　数

对数可视为指数计算的逆运算，当处理较大的数字时，对数是一种十分有效的工具。

2.5.1　对数计算

对数定义为指数计算的逆运算，若 $a = b^c$，则有 $c = \log_b(a)$，即底数为 b 时的 a 的对数。也就是说，为了得到答案 a，须计算 b 所需自乘的指数 c。

计算机中通常设置了一个或多个对数函数，例如 $\log_e()$和 $\log()$。其中，$\log_e()$常简写为 $\ln()$，即自然对数。

大多数对数计算均包含对应的指数表达式，如下所示：

- 针对任意底数，有 $\log(a)+\log(b)=\log(a\times b)$。若 $n^p = a$ 且 $n^q = b$，则 $n^{p+q} = n^p \times n^q = a \times b$。因此 $\log(a\times b)=p+q$。同时，由于 $\log(a)=p$ 且 $\log(b)=q$，因而有 $\log(a)+\log(b)=\log(a\times b)$。
- 针对任意底数，有 $k\times\log(a)=\log(a^k)$。该式源自前述运算结果，因而可直接对其加以使用。例如，假设 $n^p = a$，则有 $a^k = (np)^k = n^{p\times k}$，$\log(a^k)=p\times k$。同时，由于 $p=\log(a)$，这也意味着，$k\times\log(a) = \log(a^k)$。
- $\log_a(a) = 1$。该结果源自对数定义以及 $a^1=a$ 这一事实。
- $\log_b(a)=\dfrac{1}{\log_a(b)}$。该结果遵循前两个结论。利用 $\log_b(a)$替换前述公式中的 k 则可得到 $\log_b(a)\times\log_a(b)=\log_a(b\log_b(a))$。根据对数的定义，$b\log_b(a)$可简化为 a，因而有 $\log_b(a) \times \log_a(b) = \log_a(a) = 1$。
- $\log_a(n) \times\log_b(a) = \log_b(n)$。该结论可视为前述内容的通用结果，且源自相同的逻辑，即 $\log_a(n) \times\log_b(a) = \log_b(a\log_a(n)) = \log_b(n)$。
- 针对任意底数，有 $\log(1) = 0$。任意数字自乘 0 指数的结果均为 1（除去数字 0，该特例未经定义）。
- $\log(0)=?$。$\log(0)$未予定义。通常情况下，无法自乘数字（0 除外）某一指数次，进而得到 0 值。若读者尝试对此计算，则计算机通常会输出错误消息。

类似的情况也出现于负数的分数指数中，log 函数针对负数缺乏较好的定义，因而应尽量避免这一类计算。

2.5.2　通过对数简化计算

在便携计算器和计算机设备出现之前，对数可视为数学家、科学家以及工程师眼中的一个重要工具。例如，计算尺显示了两种对数刻度的数字集，如图 2.5 所示。

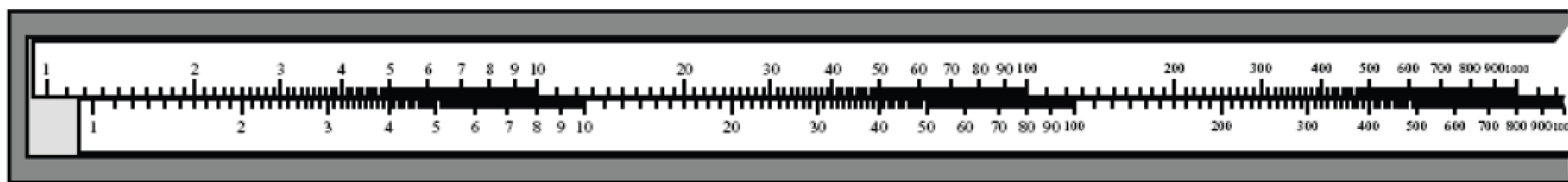


图 2.5　计算尺的下方刻度可根据上方刻度前后移动并执行计算

这里的问题是，何为对数刻度？在图 2.5 中，通过观察可知，数字并未以均匀方式分布：一

端数字较为密集，而另一端数字则相对稀疏。对数刻度的名称由来在于，若使用计算尺上某一点处的数值，并使用某一底数的对数对其进行替换，则最终结果呈线性状态。也就是说，根据当前刻度所要求的某一底数 p，计算尺 n 厘米处的 1 点表示为值 p^n。

如果尝试测量 1 和 10 之间的距离，则相当于计算 10 和 100 之间的距离。实际上，这也是对数刻度尺的主要特征。针对某一 q 值，若 $a=q\times b$ 且 $c=q\times d$，则 a 和 b 之间的距离等于 c 和 d 之间的距离。而对于线性刻度，刻度尺则对加法（而非乘法）具有相同的属性。换言之，若 $a=q+b$ 且 $c=q+d$，则 a 和 b 之间的距离等于 c 和 d 之间的距离。基本上，常规刻度尺基于加法功能，而对数计算尺则依赖于乘法运算。

上述特征较为重要，用户可通过对数计算尺方便地测算出两个数字的乘积。例如，对于 12.45 与 37.6 之间的乘法运算，可定位上方刻度的 12.45，并移动下方刻度，以使刻度 1 与其对齐。

随后，可定位下方刻度的 37.6，并查看上方刻度中的对应点。在图 2.6 中，该点处的刻度为 468.12，即两个数字间的乘积。

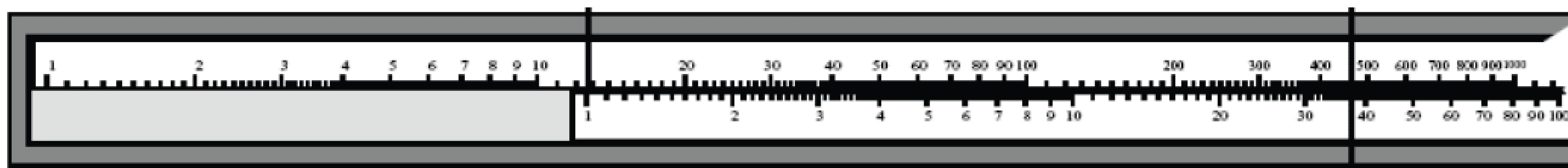


图 2.6　计算尺的操作实例

上述工作原理可描述为，1 和 37.6 之间的距离等于另一个刻度中 1×12.45 和 37.6×12.45 之间的距离。对此，读者是否已经了解了该操作与对数定义之间的关联方式（参见练习 2.3）？

2.5.3　利用对数处理大数

在当前的课堂教学中，计算尺已不再是标准的授课内容，但其后的原理依然重要。该原理表明，当处理大数时，对数可视为一种方便的工具。当使用对数时，可将较大数字限定于较小范围内。例如，数字 5000000 是 5000 的 1000 倍，而在以 10 为底数的对数运算中，二者间仅相差 3 个单位。

该描述方式较为直观，在科学计数法中，读者已经使用了这一方法。对于数字 5000000，十进制的科学计数法表示为 5e6，而 5000 则表示为 5e3。实际上，对数中的底数以及科学家说法中的进制可视为一类“标识”，进而体现了两种概念之间的相似性。对于计算尺，一类自然的处理方式则是基于乘法运算，而非加法运算。

【提示】 人类的某些器官也饰演了这一转换角色，例如，音量和音调均使用对数方式并通过大脑予以解释。中央 C 音下方的 C 音其频率也将减半，其上方 C 音的频率则加倍。同样，二者也包含“间距”一说。

对数常可有效地处理指数问题。在前述示例中，藤本植物以每星期 20%的速度增长（每个星期后，其长度值乘以 1.2），因而相关问题可描述为，多少星期后，藤本植物的长度可围绕地球一周？这里，地球的周长已知为 40000km，此处的问题可转化为：1×1.2 经多少倍后其值为 40000000m。

【提示】在古希腊时期，人们已测算出地球的周长（近似值）。埃拉托色尼首先准确地计算出了地球的周长，同时，他也是筛分法的发明者。

上述对数计算较为简单，若 $1.2^p = 40000000$，则 $p = \log_{12}(40000000)$。假设计算机可计算底数为 e 的对数，即 ln()函数，则读者需要将底数 1.2 转换为 e，如下所示：

$$\log_{1.2}(40000000) = \log_{1.2}(\mathrm{e}) \times \log_{\mathrm{e}}(40000000) = \frac{\log_{\mathrm{e}}(40000000)}{\log_{\mathrm{e}}(1.2)}$$

对应结果为 96 个星期，约为两年。相应地，可通过 power(1.2,96)对该答案进行检测，正如期望的那样，其值约为 40000000。

2.6 本章练习

【练习 2.1】尝试针对窗口内的、一段较长的文本滚动操作编写滚动栏函数集。假设存在一段文本以及显示该文本的窗口，对应函数将使用到文本的长度以及窗口的高度，进而计算既定文本位置处的滚动栏位置（反之亦然）以及滚动标记的尺寸。

【练习 2.2】试编写复利函数，该函数应可生成与前述示例类似的输出结果。对此，读者应可输入现金额度、利率以及可选的还款值，对应函数则输出各种期限的现金余额。

【练习 2.3】针对较大数值，试编写计算尺函数。该函数应遵循前述计算尺原理，并可对任意两个数字执行乘法运算。为了实现更为丰富的视觉效果，读者还可尝试加入图形界面。

2.7 本章小结

本章讨论了多种数字处理技巧，读者可将其加入至个人的数学工具箱中。第 3 章将考察另一个较为基础的数学领域，即代数运算。

至此，读者应掌握如下内容：

- 分数、比例以及百分比计算。
- 利率及其计算方式。
- mod()函数的工作原理，以及如何通过该函数计算循环数据以及整数除法。
- 因数、质数、GCD 和 LCM，以及如何使用 Euclid 算法计算两个整数的 GCD。
- 比例和比率的概念，以及如何通过二者实现数值和尺寸之间的映射。除此之外，读者还应了解某一数值范围的数值表达方式。
- 如何使用指数和对数处理大数以及乘法函数。

第 3 章　代 数 运 算

本章包含如下内容：

- 概述。
- 基本的代数运算。
- 数学方程。
- 分解并求解二次方程。
- 函数和函数图。

3.1　概　　述

本章讨论代数原理，作为数学领域中的一个分支，相关内容讨论变量的处理方式。本章首先介绍代数的基本原理，随后分析特定的计算方案，以及方程和函数的可视化技术，即函数图。

3.2　基本的代数运算

本小节回顾代数中的术语，此类术语广泛地应用于数学的各个领域中。在此基础上，本小节还将讨论数学方程等内容。

3.2.1　变量、参数和常量

代数中涵盖了两种基本的数值类型，即常量和变量。通常情况下，此类数值常用字母表示，例如 a，b，c，x，y，z。其中，常量所示内容不发生变化，而变量所定义的内容则处于变化状态。另外，常量和变量的名称与具体应用方式有关，下列内容列举了变量和常量的重要用途：

- 常量。常量值一般不发生变化，例如 e 和 φ。类似地，1 和 2 等数值也定义为常量。当临时定义字母 A 表示某一表达式时，常量可赋予任意值。
- 参数。参数定义了一个数据值，并可确定类似的数学对象集。例如，在线性方程 $y=m\times x+c$ 中，参数 m 和 c 适用于该方程的各实例中。另外，替代参数 m 和 c 的其他数据值也表达了同类信息。
- 未知数。未知数表示为未知数据项，例如，若 x 表示为某一特定数字，且有 $x+3=4$，则可计算未知数 x 的具体值。

- 变量。变量可表示为任意值，并与传递至计算机函数中的参数有几分类似。例如，读者可尝试编写函数计算数字 n 的立方根。此时，函数中的 n 即为变量。

需要注意的是，上述解释并不严格。例如，在方程 $x + 3 = 4$ 中，x 可称作未知数。另一方面，x 同时也是一个变量，并可将某一数据值代入其中（即数字 1）。通过观察可知，x 的替换值总保持相同，因而 x 也可称作常量。

针对上述术语，一种较为简单的考察方式是可变层次结构。若表达式中涉及大量的字母，例如 $u+a\times t$，则可确定一个或多个“固定”字母，而其他字母处于变化状态。其中，固定项称作参数。对于特定的参数集，可针对此类变量定义特定行为。随后，可明确地“固定”某一参数，并将其称作常量。

数学家和程序员采用类似的方式使用变量和常量，二者间的差别较为明显。程序员常采用有助于记忆的变量和常量名称，例如，程序员在程序中一般不采用 x 命名变量，相反，多使用 numOfPlayers 或 accountNumber 这一类名称。根据语言惯例，程序设计语言往往采用字符 PI 表示希腊字母 π 。通常，常量常采用首字母大写方式书写。

3.2.2 表达式和数据项

变量和常量的组合结果称作表达式，例如，$(a\times x)+(x^2\times 4)+3$ 即为由变量 a 和 x、常量 3 和 4 组成的表达式。作为指数，另一个常量为 2，x^2 表示 $x\times x$。

【提示】 如同手工计算，大多数编译器采用标准顺序计算数学表达式。首先，括号中的表达式采用递归方式计算，随后将顺序计算其他操作符，包括括号、除法和乘法以及加法和减法。因此，表达式 $1+(2\times 3-4)\times(-5+6)$ 的计算结果为 $1+2\times 1$ 或 3。另外，分数中的分子和分母则视为位于括号中。

数据项则表示为主表达式的子表达式，且不包含加法或减法计算。对此，可方便地将表达式组合为数据项。例如，在 $(a\times x)+(x^2\times 4)+3$ 中，若将 x 视为变量，a 视为常量，则表达式中存在 3 个数据项，即 $a\times x$、$x^2\times 4$ 和 3。一个数据项可包含任意数量的、变量与常量的乘积形式，即系数。在数据项 $x^2\times 4$ 中，x^2 的系数为 4。

数据项可根据变量的指数进行分类。例如，数据项 $3\times x^2$ 和 $-2\times x^2$ 视为同类项，二者中的变量 x 具有相同的指数 2。相应地，$3\times x^2$ 和 $3\times x$ 则为非同类项，皆因 x 的指数不同。

下面将对上述讨论稍作扩展，考察表达式 $2x$ 和 $p(1+q)$。在两个表达式中，多个数据彼此邻接，该结构表达了一类乘法运算。作为一种通用规则，系数通常位于变量之前（$2x$ 而非 $x2$），且位于括号外部的数据项应首先书写，即 $p(1+q)$而非 $(1+q)\ p$。另外，数据项常记为单一字符串且不包含乘号，例如，$a\times x+x^2\times 4+3$ 记为 $ax+4x^2+3$，进而可方便地根据变量指数对数据项进行合并，即 x 项、x^2 项以及数字常数项。

3.2.3 函数

函数可视为一种映射过程，从某一集合中（即定义域）获取数据，并将其转换至同一集合或

另一集合中的数值，即函数的值域。下面考察有理数与整数之间的转换行为。在数学领域中，函数常采用单一字母表示，随后是括号内的参数。例如，等式 $y = f(x)$ 将定义域中的 x 映射为值域中的 y 值。其中，函数常定义为 $f(x)$ 形式，值域中的数据值表示为定义域中的值函数。若 $f(x)$ 定义完毕，则 $f(3)$ 表示利用数值 3 替换函数 $x \to x(x+2)$中的 x 项所得到的结果，对应值为 $3\times(3+2) = 3\times5=15$（此处，"→"表示 x 映射时的逻辑符号）。

这里，数学计算与程序设计内容彼此对应。对于第 1 章中介绍的 floor()函数，可向其传入非整型数字（例如 3.45），随后，该函数返回整数 3。除此之外，读者还可使用布尔函数，此类函数将输入值映射为两个值，即 0 和 1。例如，isPrime()可视为一类较为常见的布尔函数，若输入值为质数，则函数返回 1；否则，函数返回 0。在类型化的计算机语言中，通常需要预先确定函数输入值以及输出值的类型。例如，函数需要使用到整型参数作为输入内容，并返回 double 类型数据（浮点值）。这一需求指出了函数的基本特征，即不同数据域之间的转换操作。

3.2.4　函数表达方式

大多数数学函数包含自身的符号，此类符号也体现于程序函数中。例如，"$\sqrt{\ }$"表示 sqrt()函数。另外，某些数学函数则直接对应于程序函数，例如 sin()，cos()以及 tan()。而在其他场合，这一对应关系则表现得并不明显。对于函数 x^2+2，其正规表达方式如下所示：

$$squarePlusTwo(x\colon \mathbb{R} \to \mathbb{R}) = x^2 + 2$$

其中，"$\mathbb{R} \to \mathbb{R}$"为函数正规定义中不可或缺的一部分，具体含义为，当前函数的输入和输出数据均为实数。

【提示】在上式中，由于已知函数的输出结果为正值，因而可将第 2 个"$\mathbb{R}$"替换为"$a > \mathbb{R}^+$"，即正实数集。准确地讲，函数范围位于$[2,\infty]$，即 2 至正无穷之间的全部实数集。此处，函数映射至此范围内。

3.2.5　一一对应、反函数和多值函数

对于函数 $f(x)$，若定义域内不存在两个独立的输入值 a 和 b 使得 $f(a) = f(b)$，则该函数称作一一对应。同时，此类函数存在反函数，记为 f'。在反函数中，针对各 a 值，有 $f'(f(a)) = a$。例如，3 次函数 $x \to x^3$ 即包含一一对应关系。相应地，函数 $x \to x^2$ 仅在正实数的范围内存在一一对应关系。若定义域内存在多个值可映射至值域中的同一个值，则该函数具有多对一的关系。

若值域中至少存在一个值可映射至定义域中的多个值，则此类函数称作多值函数，例如平方根函数。其中，1 和–1 均为 1 的平方根。通常，多值函数与"多对一"函数具有反向关系。严格地讲，上述函数并非真正意义上的函数。另外，针对标准的编程语言，某些自包含函数通常难以实现多值函数。例如，sqrt()表示为 $\mathbb{R}^-$～$\mathbb{R}^+$范围内的一一映射函数，且返回正平方根值。

3.2.6　多项式

多项式可视为 $x \to a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n$ 的特定函数，且 a_0，a_1，a_2，a_3 均为实数。

多项式可通过阶数加以区分，例如，由于 x 的最大指数为 1，因而函数 $x \to 2x+1$ 称作线性或一阶多项式。类似地，$x \to 2-x+3x^2$ 称作二次或二阶多项式。除此之外，多项式中还可包含多个变量，例如 $x \to x+2xy+y^2$。

【提示】尽管本小节定义了形式化函数的描述方式，但非正式形式的函数往往易于交流。因此，本书后续内容均采用形如 x^2+1 的函数，也就是说，不采用 $x \to x^2+1$ 这一类数学映射符号。

3.2.7 等式、公式和不等式

等式类似于数学表达语句，即两个表达式之间相等，例如 1=1——该式十分简单且具有相等性，例如 2+2=4。在大多数场合中，术语 “等式”包含某一变量，例如 $x+2=5$，也就是说，该等式包含一个未知项 x。若该等式为真，则可推断出 x 的具体值，即 $x=3$。相应地，这一推断过程可视为初等代数主要内容。

对此，读者应对函数、表达式以及等式予以区分。函数类似于短语并包含空白待填写内容，例如“高度值为……的……”；表达式则更进一步，并通过虚变量填充空白内容，例如“高度值为 *mmm* 的 *nnn*”；等式（方程）则通过关联关系体现表达式，例如“高度值为 *mmm* 的 *nnn* 可称作 *qqq*”。

在上述 3 个描述中，仅等式包含 true 或 false 结果。此时，由于 *nnn*、*mmm* 和 *qqq* 尚未可知，因而等式的 true 或 false 结果也处于未知状态。若对应描述为“*nnn* 等于 *nnn*”，则无论 *nnn* 如何，对应结果均为 true，此类等式称作重言式（tautology）。例如，$x+2=x+3-1$ 即为重言式——无论 x 为何值，对应结果均为 true。而等式 $x+1=2$ 则不为重言式。当 $x=1$ 时，该式为 true；当 x 为其他值时，该式则为 false。同样的情形也出现于程序中，对于变量 x 有语句“if x+2=3”，则仅当 $x=1$ 时方执行该语句后的代码，即等式为 true。相反，若已知等式为 true，则 x 值为 1。

公式（formula）可视为包含多个变量的等式，并根据其他变量定义单一变量。例如，公式 $v = u + at$ 根据 u、a 和 t 定义 v。然而，鉴于公式和等式可等同对待，因而二者仅是术语上的区分。相应地，公式常用于表达物理值之间的关系，例如距离 = 速度×时间。

不等式与等式类似，但却反映了数据间的不等性。另外，不等式的类型也多种多样，某一表达式可小于另一个表达式，例如 $x+1<x+2$。该式为重言不等式，也就是说，无论 x 为何值，该表达式总为 true。同时，表达式还可表示为一个表达式大于另一个表达式，例如 $x + 1 > y$。该表达式是否成立尚难以确定，且与 x 和 y 值有关。另一方面，不等式可简单地表达两个表达式彼此不同。在计算机语言中，表达式可记为 x != y 或 x <> y。从数学角度上讲，不等式常写为 $x \neq y$。

除此之外，还可采用混合方式表示不等式符号，例如“≤”和“≥”，分别表示“小于等于”和“大于等于”。而“≈”符号则表示为“约等于”，“≡”符号表示“恒等于”，此二者较少出现于不等式中。

3.3 等 式 计 算

通过符号操作，代数运算可实现未知项的演算，而非对其直接进行计算，本小节将回顾代数

运算中的主要方法。

3.3.1　等式配平

等式类似于一架天平，若其处于平衡状态，则两侧添加或减去相同数量，天平依然保持平衡，这一解释同样适用于等式。根据定义，等号两侧的表达式彼此相等，若两侧添加同一值，则相等性保持不变。实际上，若等式两侧加入任何非多值函数，则相等性保持不变。

这里的问题是，为何对多值函数予以限制？这里，假设存在等式 $a^2=b^2$，且对等式两边执行平方根计算，最终结果为 $a=b$。不难发现，该结果并不正确。例如，若 $a=2$，$b=-2$，则第 1 个等式成立，而第 2 个等式不成立，其原因在于，$x \rightarrow \sqrt{x}$ 为多值函数。具体而言，定义域内的两个不同值可映射为值域中的同一值。准确地讲，相应结果为 $a=\pm b$，其中，“±”表示为正负号。

等式的求解过程涉及其两侧的运算，进而求解未知项。对于函数 f 和 g，较为常见的形式是 $f(x)=g(x)$。若等式两侧减去 $g(x)$，即 $f(x)-g(x)=0$，这将产生一个新的函数 $q(x)=f(x)-g(x)$。通常情况下，等式的求解过程需要计算 0 值处的函数 q 的反函数。

针对 $2x+3=7$，该式左侧包含两项，第 1 项包含 x，第 2 项则包含了一个常量。当求解该等式时，其两侧可减去常量，如下所示：

$$2x+3=7$$
$$2x+3-3=7-3$$
$$2x=4$$

此时，等式两侧各包含一项，左侧为 $2x$，右侧为常量 4。两边除以 2 即可得到 x 值，如下所示：

$$2x=4$$
$$\frac{2x}{2}=\frac{4}{2}$$
$$x=2$$

最终结果为 $x=2$。反观上述计算过程，读者将会发现 $2\times2+3$ 等于 7。

如前所述，可采用函数方式区别对待 $2\times2+3$。对此，可定义函数 $x\rightarrow 2x+3$，进而求解其反函数。由于该函数先后乘以 2 并加 3，因此反函数可视为减 3 并除以 2，这相当于逆序计算，故反函数可表示为 $x \rightarrow (x-3)/2$。若将该函数应用于等式两侧，则等式左侧映射为 x（根据反函数的构造过程），等式右侧映射为 $(7-3)/2=4/2=2$。该处理过程与前述操作并无太大差别，但其后的理念则稍有不同。

3.3.2　简化计算

基于反函数的等式方程求解过程较为复杂，例如 $\frac{x+1}{2(x-1)-3(2-x)-x}=3x-8$。其中，$x$ 的析取操作相对困难，因而应采取适当的简化操作。经过连续的化简计算后，最后应得到等式（或函数）的最简形式。当然，最简形式与具体的计算过程有关，下列内容列举了其中的几项要点：

- 合并同类项。同类项往往包含相同的变量组合，其差别仅在于系数的不同。例如 $2x+3x=5x$，$2x$ 和 $3x$ 即为同类项，因而可合并为一项。
- 交叉乘法，该方法源自第 2 章。例如，下列表达式：

$$\frac{x}{x+1}+\frac{x+2}{x+3}$$

可简化为：

$$\frac{x(x+3)+(x+2)(x+1)}{(x+1)(x+3)}$$

该式稍显冗长，若采用 4 替换 x，则计算结果相同，如下所示：

$$\frac{4}{5}+\frac{6}{7}=\frac{4\times 7+6\times 5}{5\times 7}$$

- 移除分数。若等式一侧包含分数，等式两侧乘以分母即可移除该分数。当采用该方法时，$\frac{x}{x+1}=2$ 可表示为 $x=2(x+1)$。
- 括号展开。若数据项与括号内的表达式执行乘法运算，该项可与括号内的各项数据分别执行乘法运算，进而消除括号，相关示例如下所示：

① $2(x + 3) = 2\times x + 2\times 3=2x + 6$。

② $3x(2 - x + 4x^2) = 3x\times 2 - 3x\times x + 3x\times 4x^2=6x - 3x^2 + 12x^3$。

③ $5 - 2(2x + 1) = 5 - 2\times 2x - 2\times 1= 5 - 4x - 2= 3 - 4x$。

3.3.3 符号和置换操作

在上述示例③中，应注意第 1 个表达式中的负号，此处，括号中的各项分别乘以-2。当采用负值乘以括号中的数据值时，负号应用于括号内的全部值上。下面从不同角度考察这一问题，假设原表达式记为 $5+(-2)\times(2x+1)$，经比较后，读者可发现负号的功效。

除了负数之外，还可针对多个括号间的数据值执行乘法运算，如下所示：

$$\begin{aligned}(x-2)(2x+3)&=x\times 2x-2\times 2x+x\times 3-2\times 3\\&=2x^2-4x+3x-6\\&=2x^2-x-6\end{aligned}$$

此处可对该问题稍作扩展。也就是说，可定义一个新变量，进而将等式转换为另一形式。该方法稍显晦涩，对于某些复杂计算，该方案十分有效。例如，假设求解 $4x^4-9=0$，需要注意的是，$4x^4=(2x^2)^2$。当简化该表达式时，可新增变量 p 且有 $p=2x^2$。由于 $4x^4=p^2$，因而有 $p^2-9=0$ 或 $p^2=9$。由于 9 的平方根为 3，故 $p^2=\pm 9$。

当计算 x 时，可考察原始定义，即 $p=2x^2$，故 $2x^2=\pm 3$。由于 $2x^2$ 总为正值，因而可忽略 p 的负平方根，即 $2x^2=3$，则 $x=\pm\sqrt{\frac{3}{2}}$。如前所述，该方法可视为一种特例且较少遇到，本章稍后介绍立方值计算时将会用到这一方法。

3.3.4　对原问题进行求解

下面继续考察本节开始处的问题，该问题使用了前述内容所讨论的各种计算方法，如下所示：

$$\frac{x+1}{2(x-1)-3(2-x)-x}=3x-8$$

当求解该等式时，可首先简化其分母项，并对括号进行展开，如下所示：

$$\frac{x+1}{2x-2-6+3x-x}=3x-8$$

合并分母中的同类项，如下所示：

$$\frac{x+1}{2x+3x-x-2-6}=3x-8$$

$$\frac{x+1}{4x-8}=3x-8$$

等式两侧乘以分母即可消除分数形式，对应结果如下所示：

$$x+1=(3x-8)(4x-8)$$

展开等式右侧的括号，则可得到如下等式：

$$\begin{aligned}(x+1)&=(3x-8)\ (4x-8)\\&=12x^2-32x-24x+64\\&=12x^2-56x+64\end{aligned}$$

最后，将左侧数据项移至右侧，并再次执行简化操作，如下所示：

$$x+1=12x^2-56x+64$$

$$0=12x^2-56x+64-x-1$$

$$12x^2-57x+63=0$$

在最后一个等式中，计算顺序被调整，也就是说，将 0 调整至右侧，这也是等式的传统表达方式——若等式某一侧包含常量，通常可将其置于右侧，这也符合常见的英语表达方式，即名词置于句中的系动词之前，例如“Patch is a dog”，而非“a dog is Patch”。

3.4　分解并求解二次等式（方程）

另一种简化方案则与括号展开操作相反，即通过计算公因数的方式简化表达式。在第 2 章中曾提到，gcd()函数可应用于分母上，进而简化分数的加法运算，当前简化方案与此类似，称作因式分解，或简称为分解操作。

总体而言，当前述简化操作执行完毕，已合并同类项并消除分数形式后即可进行分解操作。针对某一等式，较好的方法是将全部数据项移至等式一侧，即生成形如 $f(x)=0$ 的等式方程。严格地讲，因式分解针对函数执行，而非等式。

若将等式确定为函数形式，则可在各数据项之间计算公共因子。例如，在表达式 $6x^2-15x+9$

中，全部数据项均包含因子 3。对此，可提取该因子并使其成为一个独立项，即 $3(2x^2-5x+3)$。

另一个例子是 x^2-4x，两项皆包含因子 x。如前所述，可通过因子 x 的乘积方式，进而分解等式左侧的表达式，对此，提取 x 并在括号中写入 $x-4$，即 $x(x-4)$。

【提示】读者可尝试对括号内外的全部数据项执行乘法运算，并查看计算结果。不难发现，对应结果与前述示例完全相同，只是此处使用了因子 x 而非数字 3。

这里，可采用函数的因式分解后的版本简化等式的求解过程。例如，针对 $3(2x^2-5x+3)=0$，等式两侧可除以 3 以得到更为简单的形式，即 $2x^2-5x+3=0$。另外，若 $x(x-4)=0$，则包含两个未知项（x 和 $x-4$），且二者的乘积为 0。如果两个数据项的乘积为 0，则可知二者皆为 0 或某一项为 0，即 $x=0$ 或 $x-4=0$，则 x 值等于 0 或 4。

3.4.1　分解示例

综上所述，下列内容显示了分解示例：

- $12x^3-8x^2$ 包含公共因子 $4x^2$，其中，4 表示为 12 和 8 的因子，x^2 为自身和 x^3 的因子，因而可提取出该因子，进而生成 $4x^2(3x+2)$。
- $-2x-3x^2$ 包含公共因子$-x$。由于负数除以负数的结果为正数，故因式分解后的结果为 $-x(2+3x)$。
- $2xy^2z+4x^2yz$ 包含公共因子 $2xyz$，经因式分解后，最终结果为 $2xyz(y+2x)$。
- $\frac{x}{2}-\frac{x^2}{4}$ 包含公共因子 $\frac{x}{4}$。该过程可划分为两个步骤：步骤一涉及化简操作，即 $\frac{x}{2}-\frac{x^2}{4}=\frac{2x-x^2}{4}$。待进一步分解后，步骤二将生成 $\frac{2x-x^2}{4}=\frac{x}{4}(2-x)$。

除了简单项可作为公共因子之外，表达式也可视为因子。例如，在表达式 $x(x+1)+3(x+1)$中，表达式$(x+1)$即为公共因子，因而原表达式可分解为$(x+1)(x+3)$。

3.4.2　因子和二次表达式

读者可尝试对$(x+1)$ $(x+3)$进行括号展开、合并同类项并观察计算结果——最终结果为 x^2+4x+3。当分解该表达式时，鉴于前述乘法运算生成了当前结果，因而可逆序处理并得到相应的分解式。当然，此类表达式（例如 x^2+4x+3）难以直接分解为形如 $x(x+1)+3(x+1)$的计算结果。

表达式 x^2+4x+3 称作二次表达式，而二次表达式可定义为形如 ax^2+bx+c 的表达式。若某一二次表达式可分解，对应结果为$(px+n)(qx+m)$。

【提示】在形如 ax^2+bx+c 和$(px+n)(qx+m)$的通用表达项中，a，b，p 和 q 可视为系数符号，除非其值大于 1，否则对应系数通常不予显示。例如，表达式一般不写为 $1x^2+1x+2$（其中，c 表示为常量且等于 2）。

为了构造二次等式（方程）的通用化简模式，可通过 $ax^2+bx+c=(px+n)(qx+m)$令两个表

达式相等。

若对等式右侧展开并合并同类项，则可生成如下形式：

$$\begin{aligned} ax^2 + bx + c &= px^2 + pmx + nx + nm \\ &= px^2 + (pm + n)x + nm \end{aligned}$$

其中，x 为变量，a，b 和 c 为参数。针对任意 x 值，若给定 a，b 和 c，则当前等式可能为 true。此时，各个 x 项之间彼此匹配，若假设 $p = 1$，则推论过程如下所示：

```
If ax2 = px2, then a = p.
Likewise, if (pm + n)x = bx, then b = am + n (since a = p).
Therefore, nm = c.
```

下面考察 a=1 时的情形，这里，二次表达式的通用形式为 $x^2 + bx + c$。经适当调整后，有 $n + m = b$ 和 nm=c。进一步讲，n 和 m 表示为两个数字，二者乘积为 c，其和为 b。

这里的问题是，如何获取此类数字？一种方案是采用验算法，相关示例如下所示：

- 针对 $x^2 + 4x + 3$，由于 $b = 4$ 且 $c = 3$，因而对应数字的乘积为 3，其和为 4，据此，相关数字为 1 和 3，表达式可分解为$(x + 1)(x + 3)$，这与前述结果完全相同。
- 针对 $x^2 + 3x - 4$，由于 $b = 3$ 且 $c = -4$，因而对应数字的乘积为−4，其和为 3，而数字 4 和−1 可满足当前描述，表达式可分解为$(x + 4)(x - 1)$。
- 针对 $x^2 - 5x + 6$，由于 $b = -5$ 且 $c = 6$，因而对应数字的乘积为 6，其和为−5，数字−2 和−3 可满足当前描述，当前表达式可分解为$(x - 2)(x - 3)$。
- 针对 $x^2 - 5x - 6$，由于 $b = -5$ 且 $c = -6$，因而对应数字的乘积为−6，其和为−5，数字−6 和 1 可满足当前描述，当前表达式可分解为$(x - 6)(x + 1)$。

需要注意的是，读者寻找的数字可根据 b 和 c 的符号产生变化。在最后两个示例中，b 和 c 的绝对值相同，但最终结果并不一致。

对此，是否可得到两个数字 n 和 m 并满足上述需求条件？答案是否定的，例如，若 b=1 且 c=1，则不存在相关数字可满足二者之和以及乘积皆为 1。作为一类通用规则，仅当 $b^2 \geqslant 4ac$，方可对二次表达式进行分解。

若给定二次表达式 ax^2+bx+c，若 a 不等于 1，情况又当如何？其推论过程如下所示：

$$b = am+n$$

$$nm = c\text{，则有 } amn = ac$$

上述过程计算两个数值 am 和 n，其和为 b 且二者乘积为 ac。该方法并无太多变化，待 am 计算完毕后，除以 a 即可得到 m。

这里，考察前期示例 $12x^2 - 57x + 63 = 0$。为了分解该式，等式两侧可除以公共因子 3，进而生成 $4x^2-19x+21$。对此，有 b=−19 且 ac=4×21=84。最终，两个数字的乘积为 84，和为−19，即−7 和−12。此处，可将二者分别命名为 n 和 am。由于 $a = 4$，且 4 为 12 的因子，因而可选取 $n = -7$ 且 m=−3。通过该方式，此处无须使用到分数。

通过上述内容，当前结果为 $4x^2-19x+21=(4x-7)(x-3)$。鉴于二者乘积为 0，因而有 $x-3=0$ 或 $4x-7=0$。据此，可得到 x=3 或 x=7/4。为了验证这一结果，可将任一值代入至原等式中，以查看其值是否为 0。

求解二次等式较为常见的方式是二次公式，即 $x=\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$。当与该式协同工作时，需要注意的是，若 $b^2<4ac$，则根号值的符号为负。此时，x 不存在实根；若 $b^2>4ac$，则 x 存在两个值，且分别对应于正根和负根；若 $b^2=4ac$，则 x 仅存在一个解，例如 $x^2-6x+9=0$，该式可分解为$(x-3)(x-3)$或$(x-3)^2$。

除此之外，另一种二次表达式涉及平方差，例如 x^2-25，该式可分解为$(x-5)(x+5)$，其分解方式提供了一种稳定的因式分解模式。如果读者在两个相差较小的数字之间执行乘法运算，则可计算两个数字均值的平方，并减去半差值的平方，如下所示：

$$
\begin{aligned}
202*198 &= (200+2)*(200-2) \\
&= 200^2-2^2 \\
&= 40000-4 \\
&= 39996
\end{aligned}
$$

3.4.3　求解 3 次等式

高阶多项式涉及较多的计算，例如 3 次函数。3 次函数（或 3 次等式）通常至少包含一个根（例如值 x 且满足 $f(x)=0$），且最高可达 3 个根。标准的 3 次函数形如 $Ax^3+Bx^2+Cx+D=0$，其中，D 项称作判别式。

类似于二次等式，可通过计算具有简单形式的等式，进而得到 3 次等式的解。对于初学者而言，可借鉴前述二次等式的求解方案，例如，将等式除以 A 以获得简化形式 $x^3+ax^2+bx+c=0$，随后，利用新变量 t 替换变量 x，且有 $t=x+\dfrac{a}{3}$，这将生成不包含二次项的新 3 次等式，如下所示：

$$t^3+3pt+2q=0$$

其中，$p=\dfrac{b}{3}-\dfrac{a^2}{9}$，$q=\dfrac{a^3}{27}-\dfrac{ab}{6}+\dfrac{c}{2}$。

如前所述，此处关键之处在于判别式 $D=p^3+q^2$，如下所示：

- 若 $D>0$，则等式包含一个实根 $r+s$，其中 $r=\sqrt[3]{-q+\sqrt{D}}$，$s=\sqrt[3]{-q-\sqrt{D}}$。
- 若 $D=0$，则等式包含两个根 $2\times\sqrt[3]{-q}$ 和 $-\sqrt[3]{-q}$。
- 若 $D<0$，则可通过三角函数计算 3 个根。对应 t 值如下所示：

$$t=2\sqrt{-p}\cos\theta$$

其中，$\theta=\dfrac{1}{3}(\cos^{-1})(\dfrac{-q}{\sqrt{-p^3}})(-2\pi k)$，且有 $k=0$，1 或-1。

待 t 值计算完毕后，则可在此通过 $x=t-\dfrac{a}{3}$ 求解 x 的值。

不难发现，该过程较为复杂。另外一方面，该过程也构成了相关算法的基础内容，并可实现较为稳定的程序函数，对应伪代码如下所示：

```
function solveCubic(a,b,c,d)
  //d is the coefficient of the cubic term, the default being 1
```

```
    if d is defined then divide a, b and c by d
      set p to b/3 - a*a/9
      set q to a*a*a/27—a*b/6 + c/2
      set disc to p*p*p + q*q
      if disc>=0 then
        set r to cubeRoot(-q+sqrt(disc))
        if disc=0 then set ret to [2*r, -r]
      otherwise
        set s to cubeRoot(-q-sqrt(disc))
        set ret to [r+s]
      end if
    otherwise
    set ang to acos(-q/sqrt(-p*p*p))
    set r to 2*sqrt(-p)
    set ret to an empty array
    repeat for k=-1 to 1
      set theta to (ang—2*pi*k)/3
      append r*cos(theta) to ret
    end repeat
    end if
    subtract a/3 from each element of ret
    return ret
  end function
```

【提示】该算法源自 Eric Lengyel 所编写的《*Mathematics for 3D Game Programming*》一书，其实现过程较为优雅。网络上的其他方法并无太多新奇之处，且往往会涉及某些复数知识。

3.4.4　求解联立方程

通常存在两种方法求解等式（方程），方法一为替换法，方法二则是消去法。尽管两种方法可彼此互换，但具体实现还将取决于个人喜好。

3.4.5　替换法求解联立方程

若某一等式中包含两个未知项，通常无法得到最终结果。例如，若 $x + y = 5$，则 x 可能为 1 且 y 为 4；或者 x 为 10 且 y 为−5，此类组合趋于无穷。然而，待信息得到进一步丰富后，则可对未知项求解。总体而言，若包含 n 个未知项，则需要 n 个独立方程对此予以计算（这里，独立方程是指，该方程无法通过初始方程推导得出）。若方程数量少于 n，则求解过程缺乏足够的信息；相反，若方程数量大于 n，则求解过程存在冗余信息。无论如何，方程均无法获得一致的计算结果。

包含相同未知项的等式集合称作联立方程组，下面首先讨论包含两个未知项的联立方程组，如下所示：

$$x + 3y = 10$$

$$5x - 2y = -1$$

对此，存在多种方法求解上述方程组，此处首先介绍替换法。当采用替换法时，可通过某一方程计算未知值（作为另一个方程的函数），并于随后将该值代入第 2 个方程中。此处，若重构方程 1，则可得到 $x = 10-3y$，若将该项代入方程 2 中，则可得到如下结果：

$$5(10 - 3y) - 2y = -1$$

$$50 - 15y - 2y = -1$$

$$-17y = -51$$

$$y = 3$$

上述处理过程消除了 x 项。通常，可通过某一方程生成基于 y 且不包含 x 的新方程。当 y 值计算完毕后，可使用 x 函数进行计算，即 $x=10-3y=10-9=1$。若将 x 和 y 值代入方程 2 中，则可得到 $5x-2y=5-6=-1$。

当处理非线性方程时，替换法十分有效。非线性方程多包含形如 x^2、y^2 以及 xy 的数据项。为了进一步讨论替换法，考察下列等式：

$$3x + 2xy = 7$$

$$2x + 5y - y^2 = 8$$

等式 1 可分解为 $x(3+2y)=7$，即 $x=\dfrac{7}{3+2y}$。随后，可将该值替换至等式 2 中，对应 3 次等式如下所示：

$$2\left(\frac{7}{3+2y}\right) + 5y - y^2 = 8$$

$$\frac{14 + 5y(3+2y) - y^2(3+2y)}{3+2y} = 8$$

$$14 + 5y(3+2y) - y^2(3+2y) = 8(3+2y)$$

$$14 + 15y + 10y^2 - 3y^2 - 2y^3 = 24 + 16y$$

$$0 = 2y^3 - 7y^2 + y + 10$$

针对 3 次等式，可将–1 代入右侧表达式中，最终结果为 0。这也意味着，表达式$(y+1)$为因子（这也是一般性定理的一个实例，若 a 表示为函数 $f(x)$ 的根，则$(x–1)$为该函数的因子。回忆一下，根值可表示为 $f(a)=0$）。

若知晓某一因子，则表达式的分解过程相对简单，只需一次执行单一步骤即可。针对上述等式 $2y^3-7y^2+y+10$，若某一因子已知，则当前问题可描述为 $2y^3-7y^2+y+10=(y+1)(\cdots)$。随后，需要计算括号内的表达式。针对这一目标，括号内的首项需为 $2y^2$，扩展后首项则变为 $2y^3$，这意味着，因子可描述为$(y+1)(2y^2+\cdots)$。

若尝试扩展这一具有试验性质的临时表达式，则可得到 $2y^3+2y^2+\cdots$，但 y^2 中的数据项实际为$-7y^2$。相应地，还需要另一项$-9y^2$ 以构成差运算。也就是说，当前因式分解式为$(y+1)(2y^2-9y+\cdots)$。再次强调，扩展上述表达式时，当前结果为 $2y^3-7y^2+9y+\cdots$。此时，须匹配 y 的系数，即 1。为了获得这一结果，需要向当前答案中加入 $10y$，因而最终分解结果为$(y+1)(2y^2-9y+10)$。当扩展括号内的数据项时，即可得到对应结果。

针对当前二次函数，可采用前述方法对其执行分解操作。对此，需要两个数字，其乘积为

20，二者之和为–9。不难发现，对应数值为–9 和–4，因而对应因式表示为$(y+1)(y-2)(2y-5)$。

从前述 3 次等式的因式分解过程可知，y 可能包含 3 个有效值，即 $y=-1$，$y=2$ 或 $y=5/2$。针对各值，可在等式 1 中进行替换，进而得到 3 个(x,y)值对，即(7,–1)，(1,2)或(7/8, 5/2)。相应地，将任意值对代入等式 2 中，即可得到答案 8。

上述处理过程较为繁琐，皆因对应问题相对复杂，同时，这也体现了替换法的应用价值。

3.4.6 基于消去法的联立方程

除了替换法之外，联立方程还可通过消去法求解。为了介绍消去法的工作方式，下面考察包含两个变量的线性联立方程：

$$3x+2y=2$$
$$2x+5y=16$$

尽管该方程组可通过替换法求解，但还可增加各方程的倍数。若已知 $a=b$ 且 $c=d$，则可计算方程的线性和，例如 $2a+3c=2b+3d$，进而可消除特定的变量。

相应地，若第 1 个方程乘以 2，则有：

$$6x+4y=4$$

第 2 个方程乘以 3 则有：

$$6x+15y=48$$

若从第 2 个方程中减去第 1 个方程，则可得到关于 y 的方程，对应结果如下所示：

$$6x+15y-6x-4y=48-4$$
$$11y=44$$
$$y=4$$

随后，可将该值代入方程 1 中，并可获得如下结果：

$$3x+2\times4=2$$
$$3x=-6$$
$$x=-2$$

这里的问题是，如何确定两个方程之间的乘数？该过程与公分母的计算相同。若消去变量 x，则需要计算两个方程中 x 的公分母。这里，3 和 2 的公分母为 6。正如分数那样，随后各方程可乘以公分母除以 x 系数后的数值。

为了对此予以进一步说明，下面考察另一个示例，如下所示：

$$3x+10y=2$$
$$5x+6y=14$$

该示例将消除方程中的 y 项。这里，方程中的 y 系数分别为 10 和 6。已知 10 和 6 的最小公倍数为 30，因而方程 1 乘以$\frac{30}{10}=3$，方程 2 乘以$\frac{30}{6}=5$，对应的新方程如下所示：

$$9x+30y=6$$
$$25x+30y=70$$

从方程 4 中减去方程 3，则可得到下列等式：

$$25x-9x=70-6$$

$$16x=64$$

$$x=4$$

将该值代入方程 1 中，则可得到如下结果：

$$3\times4+10y=2$$

$$12+10y=2$$

$$10y=-10$$

$$y=-1$$

将两个值代入方程 2 中并进行检测，如下所示：

$$5\times4+6\times(-1)=20-6=14$$

3.4.7　方程组求解函数

消除法可用于求解任意线性联立方程组，作为演示，一类方案是获取相关算法，并编写对应函数实现消去法。

假设存在 n 个线性方程并包含 n 个变量，对此，可将各方程记为包含 $n+1$ 个元素的数组。例如 $2x+3y=3$ 对应于[2,3,3]。随后，全部方程组可表示为 n 个此类数组的数组，并可通过函数参数 simul 表示，对应函数如下所示：

```
function solveSimultaneous(simul)
set redux to an empty array
set n to the number of elements of simul
repeat for i=n down to 1
repeat for j=i down to 1
if simul[j][i] is not 0 then
set row=simul[j] and quit this loop
end if
end repeat
if no row found then return "no unique solution"
divide row by row[i]
add row to redux
delete row from simul
repeat for j=i-1 down to 1
if simul[j][i] is not 0 then
subtract row*simul[j][i] from simul[j]
end if
end repeat
end repeat
set output to an array with n elements
repeat for i=n down to 1
set sum to 0
repeat for j=i+1 to n
add redux[i][j]*output[j] to sum
end repeat
```

```
set output[i] to redux[i][n+1]-sum
end repeat
return output
end
```

solveSimultaneous()函数涉及的算法包含两部分内容。首先，函数逐一考察各方程，并获取包含目标系数的方程。若方程包含 3 个变量且当前变量为 x，例如 $2x+4y+z=8$，该式除以 x 的系数后可得到 $x+2y+\frac{z}{2}=4$。其中，x 的系数为 1。最终，结果方程加入至简化方程列表中（该列表称作 redux）。随后，可从全部剩余方程中减去对应倍数的简化方程，进而消除各方程中的 x 项。

假设某一个方程为 $3x-z=5$，则需要从中减去 3 倍的简化方程，其结果为$-6y-\frac{5z}{2}=-7$。该步骤从方程中消除了 x 项，但也同时引入了 y 项——这并非是问题，当前目标仅是消除 x 项。在处理过程的最后阶段，redux 中某一方程的 x 系数为 1，其他方程的 x 的系数均为 0。

若任何阶段无法获取基于当前变量的、非 0 系数方程，则处理过程停止，且 n 个方程处于非独立状态。此时，存在两种可能性：方程无解，抑或方程存在无穷多个解。

当针对 y 和 z 重复上述过程时，redux 中的方程几何包含下列特征：

- 针对前$(i-1)$个变量，第 i 个方程包含 0 系数。
- 针对第 i 个变量，第 i 个方程包含系数 1。

处理过程的第 2 个阶段将使用上述信息求解方程，对应结果源自逆向计算。需要注意的是，最终结果通常较为简单，即最后一个变量值，例如 $z=2$，因而可将其写入至输出结果中。当前，可查看倒数第 2 个方程，例如 $y-2z=3$。由于 z 值已知，因而可将其代入至方程中，进而快速求解 y 值。此处需要执行方程的逆向计算，替代后续变量并求解当前未知项。

联立方程多出现于物理学中，尤其是碰撞检测计算。尽管这一话题可进行多方扩展，而当前阶段的主要问题则是探讨函数行为的可视化结果。

3.5　函数和函数图

本小节讨论函数及其应用方式，其中一个极为重要的应用即是函数的可视化效果，而函数图可视为此类效果的主要形式。

3.5.1　何为函数图

函数图体现了视觉化的数据表达，其标准形式为二维笛卡儿图，即有序数值对以点的形式绘制于笛卡儿平面上。其中，笛卡儿平面采用一个圆的和两个坐标轴加以定义。这里，原点表示为(0,0)数据点，坐标轴则是平面内的某一方向，并采用通过原点的一条直线表示，其箭头表明了对应方向。另外，分布于轴向上的数据值可为负值或正值。具体而言，位于原点上方或右侧的数值为正值，而箭头则表明，全部方向上的数据值趋于无穷大，如图 3.1 所示。

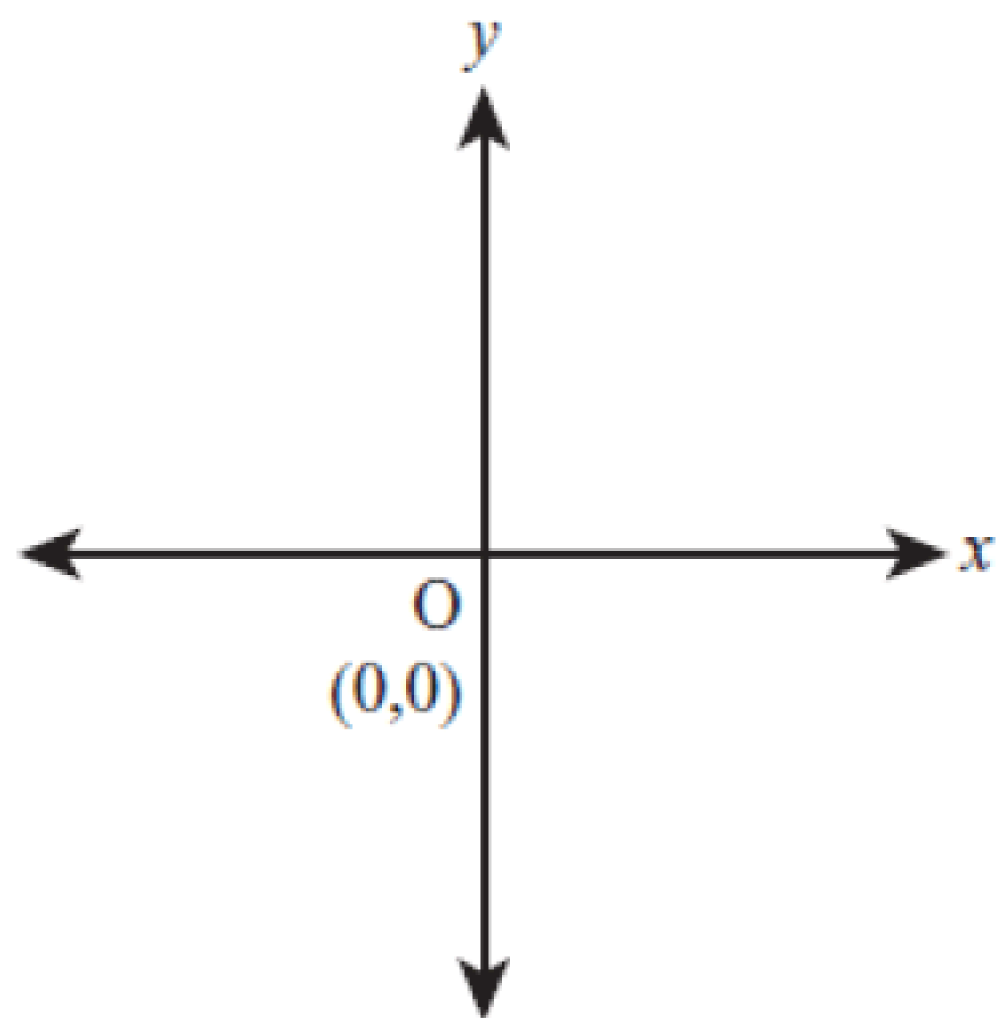


图 3.1　笛卡儿平面

针对表示数值对的笛卡儿平面，各轴须对应于数据对中的某一数字。其中，水平轴表示第 1 个数字，垂直轴则表示第 2 个数字。两个轴向通常标记为 x 和 y，且二者彼此垂直。如图 3.2 所示，若各轴向上标有刻度，则数值对(a,b)可表示为第 1 个轴向上的距离 a，以及第 2 个轴向上的距离 b。该过程称作点(a,b)的标绘，数值 a 和 b 则称作坐标。若轴向分别表示为 x 和 y，则 a 定义为 x 坐标，b 定义为 y 坐标。

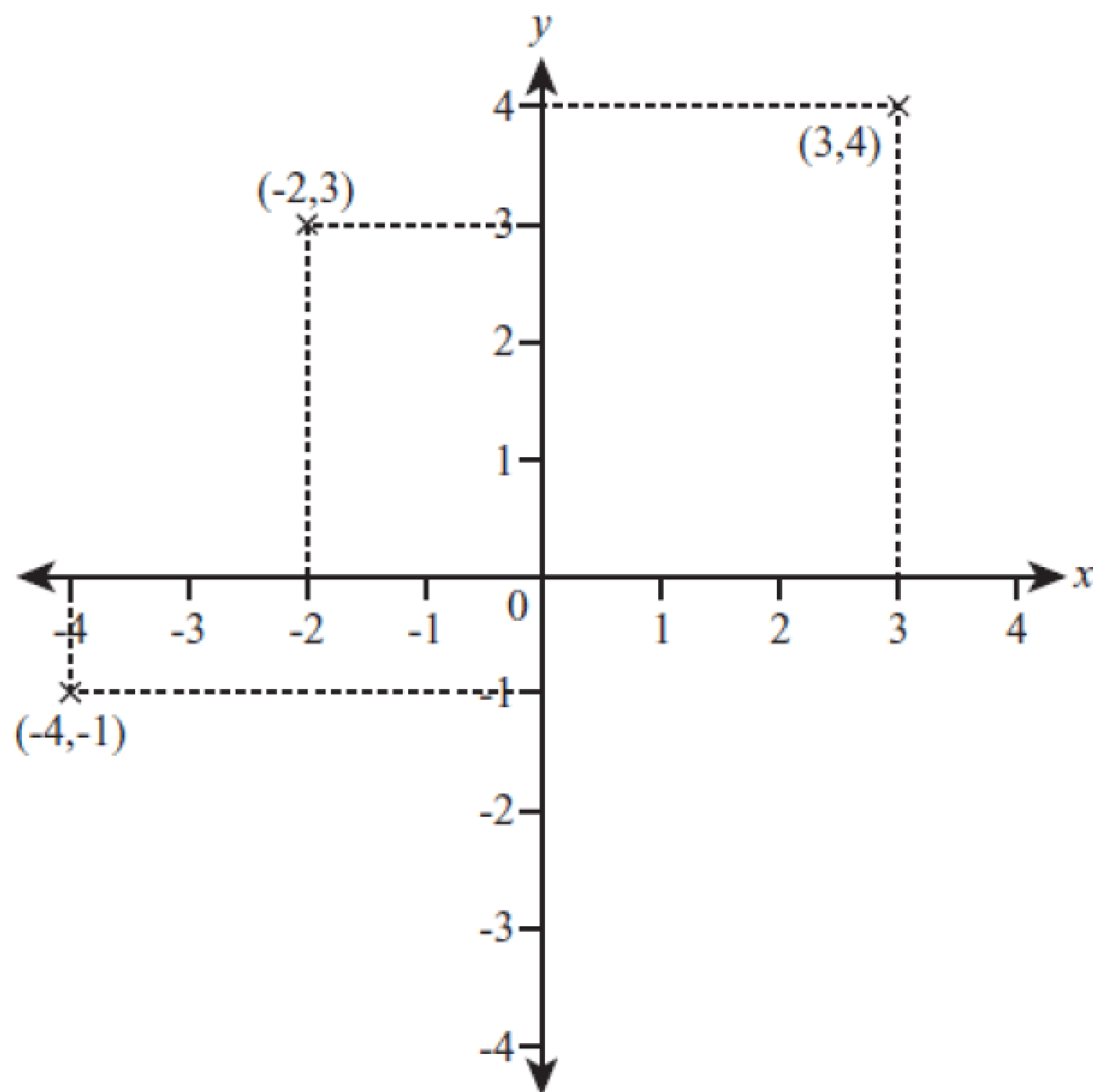


图 3.2　绘制于笛卡儿平面上的数据点(3,4)，(−2,3)和(−4,−1)

另外，读者还可采用逆向处理过程，即读取平面上的点 P 值。若垂直于 y 轴绘制一条经过点 P 的直线，则与 y 轴相交于点 Q 处。如图 3.3 所示，若在 x 方向测量 Q 与 P 之间的距离，这将生成 P 的 x 坐标；若测量 y 方向上的原点（通常简写为 O）与 Q 之间的距离，则生成 P 的 y 坐标。在图 3.3 中，点 P 被标记，其坐标为(2, 4)。

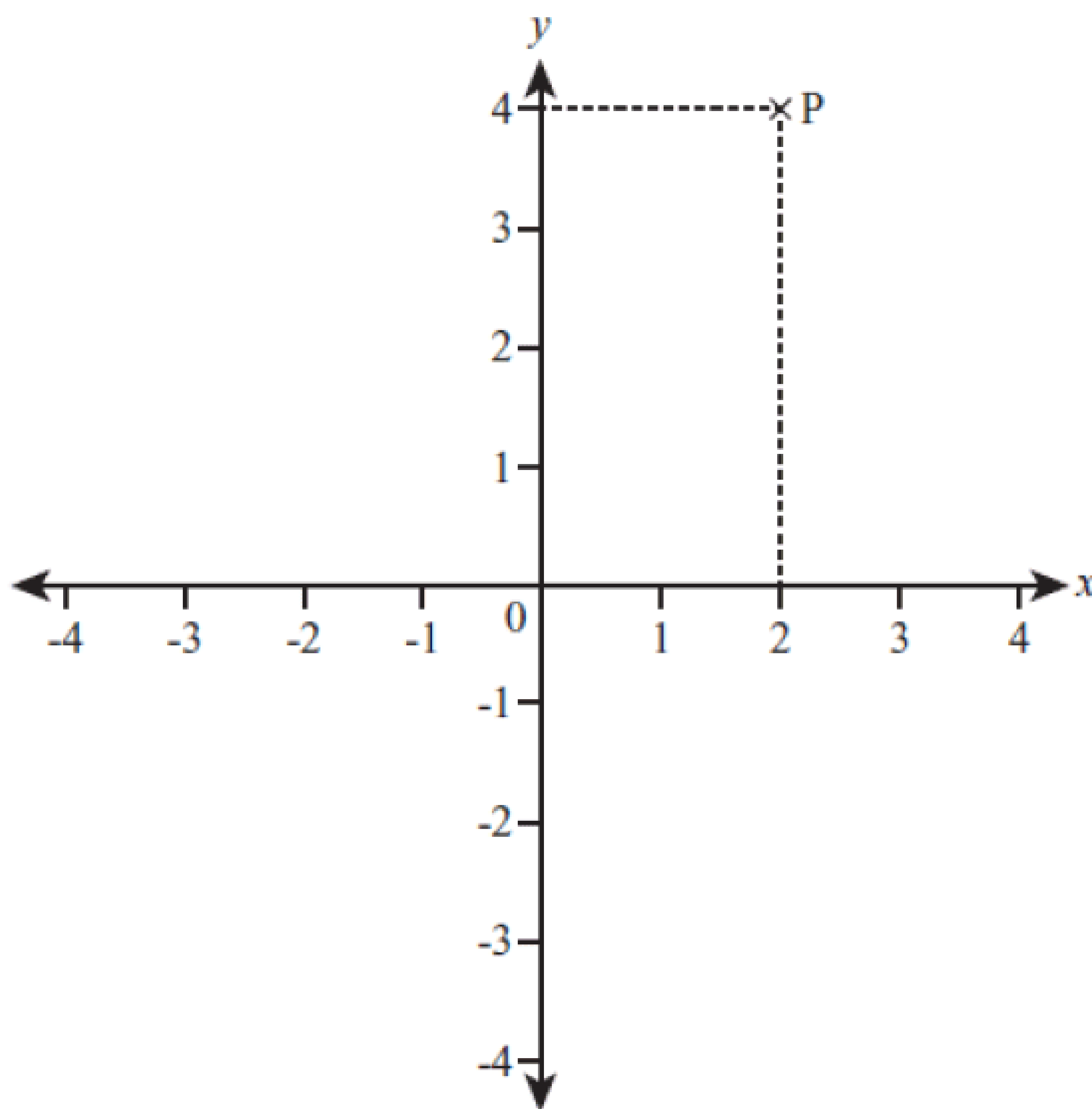


图 3.3　根据函数图读取数据点(2,4)

3.5.2　函数图的绘制和检测

图像可有效地表达函数的行为。对于函数 $f(x)$，可通过 x 值计算 $f(x)$，绘制点 $(x, f(x))$，进而以图形方式描述该函数。通常情况下，可沿水平轴向绘制变量，并沿垂直轴向绘制函数的输出结果，即根据 x 绘制 $f(x)$。除此之外，还可通过 y 标记垂直轴向，并绘制 $y = f(x)$ 的函数图。例如，若 $f(x) = 2x + 1$，则绘制 y=2x+1 的函数图。

【提示】本章仅讨论单变量函数，而多变量函数的绘制过程较为复杂。例如，若绘制包含两个变量的函数，则需要使用到三维图形（或表面），MATLAB 等工具可方便地实现三维图形的绘制工作。当绘制包含 4 个或 5 个变量的函数时，则会使用到四维或更多维图形，此时，数学软件极为有用。

计算机的图形绘制过程较为直观，其细节内容取决于既定程序设计语言提供的类和函数。例如，drawGraph()函数根据传入的参数和 x 的范围绘制函数图。当表现所绘制的、均匀分布的 x 的数量时，参数 resolution 用于确定函数图的准确性。另外，绘制过程自动标定 y 轴，以使其以整幅函数图匹配。同时，函数图的维度尺寸作为第 2 个参数予以传递，且函数图的绘制通常包含两个轴向。

【提示】大多数程序设计语言的图形函数将屏幕的左上角作为点(0,0)，并据此向右下方进行绘制。该方案有别于常见的图形绘制过程。drawGraph()函数暂且忽略此类细节内容，该函数仅标定一点或绘制一条直线。为了实现相对于坐标轴原点的直线和图像标定操作，需将数据值转换至坐标平面内。

drawGraph()函数的伪代码如下所示：

```
function drawGraph(functionToDraw, minX, maxX, resolution, width, height)
  //calculate values of the function
  set xValues to an empty array
  set yValues to an empty array
  set spacing to (maxX—minX) / (resolution - 1)
  //spacing is the distance between consecutive x values
  repeat for i = 0 to (resolution—1)
    set x to minX + i * spacing
    set y to calculateValue(functionToDraw(x))
    //how this is done depends on how you want to represent
    //the function. In the version on the CD-ROM, you can
    //pass either a string such as "x*x + 2*x + 3" or the
    //name of any function defined somewhere. The latter is
    //more flexible as it allows you to deal with special
    //cases such as "undefined" (see later in the function)
    append x to xValues
    append y to yValues
  end repeat

  //calculate the scale of the graph
  set leftX to min(minX, 0)
  set rightX to max(maxX, 0)
  //leftX and rightX are the x-values at each end of the
  //x-axis to be drawn
  set xScale to width / (rightX - leftX)
  set topY to max(largest(yValues), 0)
  set bottomY to min(smallest(yValues) , 0)
  //largest() and smallest() should return the largest and
  //smallest values in the array respectively
  set yScale to height / (topY—bottomY)

  //draw axes
  set x0 to xScale * (-leftX)
  set y0 to yScale * (-bottomY)
  //x0 and y0 are the positions within the graph of the axes
  draw a line from the point (x0, 0) to the point (x0, height)
  //this is the y-axis—you should also add arrows,
  //a scale and labels here
  draw a line from the point (0, y0) to the point (width, y0)
  //this is the x-axis

  //draw the function
  set currentPoint to 0
  repeat for i = 1 to the number of elements in xValues
    set x to xValues[i]
    set y to yValues[i]
    if y = "undefined" then
        set currentPoint to 0
      otherwise
      set thisPoint to ((x—leftX)* xScale, (y-ybottom) * yScale)
      if currentPoint = 0 then
```

```
        plot the point thisPoint
        otherwise
          draw a line from currentPoint to thisPoint
      end if
      set currentPoint to thisPoint
    end if
  end repeat
end
```

图 3.4 显示了 drawGraph()函数生成的样本数据，该版本函数提供了绘制数据的标定功能，读者可访问本书的合作网站以获取对应的源代码。

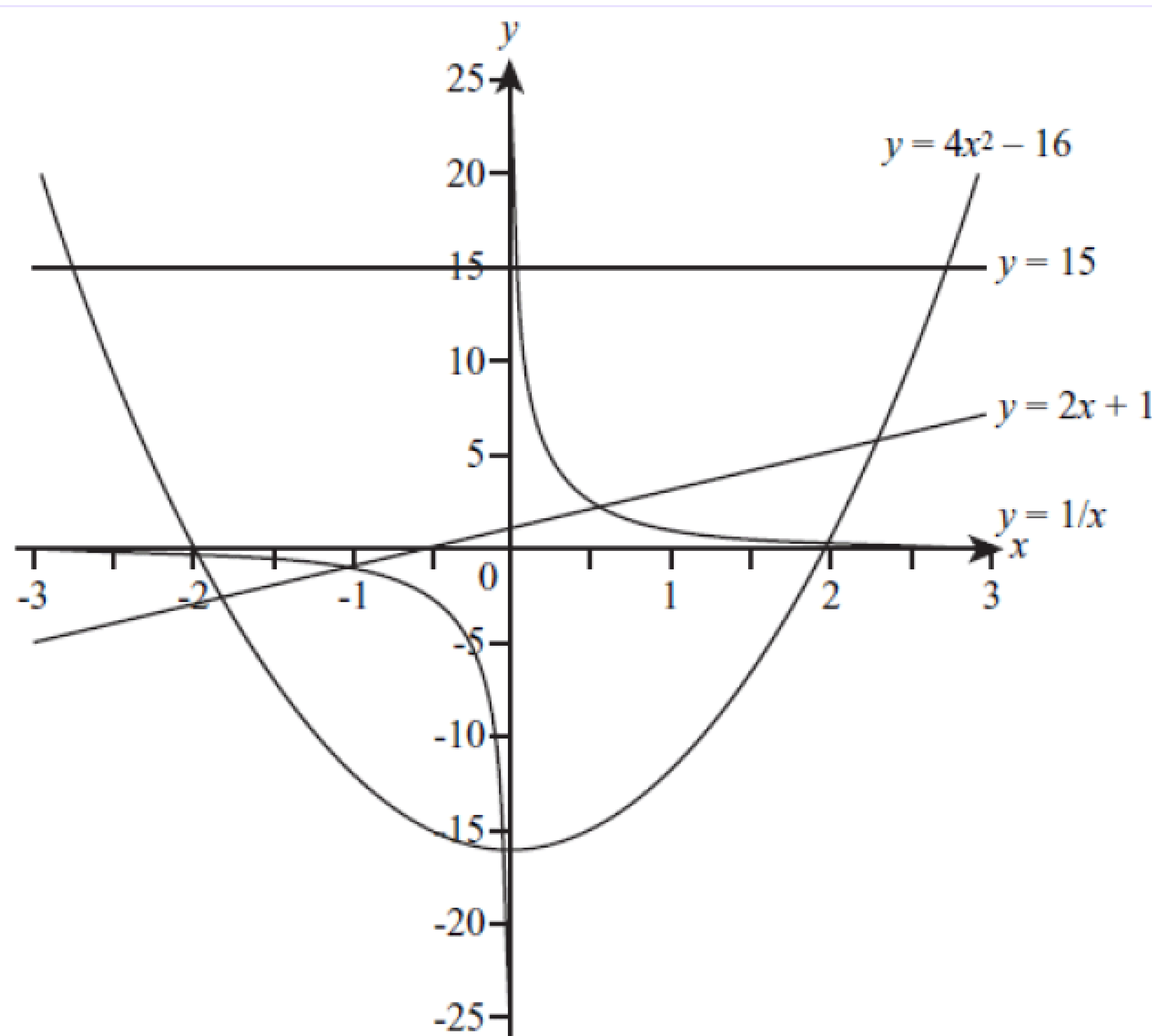


图 3.4　函数 $y = 15$，$y = 2x + 1$，$y = 4x^2 - 16$ 以及 $y = \dfrac{1}{x}$ 的函数图

尽管 drawGraph()函数生成了较为基本的数据，但作为一类模板以及图函数，其行为依然值得借鉴，相关特征如下所示：

- 水平直线体现了方程 y=15，该方程基于常量函数 $f(x)=15$。也就是说，无论 x 值如何，函数均返回 15。读者还可根据 $x = c$ 绘制一条垂直线，且 y 轴沿直线 x=0 进行绘制。
- 对角线体现了方程 y =2x+1，同时也说明了 x 项和常数项构成的函数的线性特征。在函数图中，此类函数均呈现为一条直线。
- U 形曲线称作抛物线并体现了 $y=4x^2-16$ 方程。需要说明的是，全部二次函数均会生成类似的形状，若 x^2 项为负，则曲线逆置。
- $y=\dfrac{1}{x}$ 的函数图逼近坐标轴，在 x 逐渐靠近 0 值的过程中，$\dfrac{1}{x}$ 值也随之增加，即趋于无穷大。类似地，若 x 值趋于无穷大，则 y 值随之减小，但不会为 0。此处，x=0 和 y=0 称作函数的渐近线。

3.5.3 函数图反映的数据

尽管函数图并未包含与函数相关的新增信息，但此类图像在数学、科学以及科技领域中十分重要，并可快速地反映出某些较为重要的数据片段，例如直线于何处与坐标轴相交？是否达到最大值或最小值？是否趋于无穷？更为重要的是，读者可据此推导出与当前函数相关的其他信息。为了对此予以进一步说明，下面再次考察图 3.4。

1. 水平函数

横向水平直线包含两个主要特征，其水平特征反映了对应函数为常量函数，且与 x 值无关；其次，函数在 y=3 处与 x 轴相交。上述两个事实体现了函数的全部特征，作为一类常量函数，其行为均十分类似。

2. 对角线函数

对角直线可通过多种方式描述，其中两种较为重要的方法是斜率法和截距法。直线的斜率可采用与山峰梯度相同的方式加以定义，其中，垂直距离与水平距离之间的比值称作斜率（梯度）。针对直线上的两点，斜率计算可描述为：二者间的垂直距离除以水平距离。此时，直线穿越(2, 5)和(–0.5, 0)两点，即垂直方向上的 5 个单位除以水平方向上的 2.5 个单位，最终结果为 2。

截距与直线穿越 y 轴时的交点有关，在当前示例中，y =1。在方程 y =2x+1 中，不难发现，斜率为 2，截距为 1。对于直线而言，这一结论较为常见，即斜率为 x 的系数，而常量表示为截距。根据这一特征，线性方程可表示为 $y=mx+c$。

3. 抛物线函数

与水平直线和对角直线相比，抛物线则提供了更多信息。首先，该曲线可呈递减之势，例如碗状曲线，或者递增之势，又如尖峰曲线。这也意味着，x^2 项的符号发生变化，具体而言，碗状曲线的 x^2 系数为正值，而尖峰状曲线的 x^2 项系数为负数。

其次，通过观察抛物线与 x 轴的交点可知函数的根值状态（回忆一下，若 $f(x)$ 的根值为 a，则满足 $f(a)=0$）。此时，曲线分别在+2 和–2 处穿越 x 轴，即函数的根值为+2 和–2。这也表明，(x+2)和(x–2)为该函数的因式。若抛物线仅与 x 轴相切，则函数包含单一根值，当前函数为完全平方函数。若曲线未与 x 轴相交，则该函数不包含实根且无法实现因式分解。

最后，类似于直线，可根据曲线与 y 轴的交点确定函数的常量。除此之外，还可根据曲线定义函数的最大值和最小值，即定位抛物线的折回点并确定该点的 y 值。

4. 渐近线函数

函数 $\frac{1}{x}$ 将生成渐近线图像，此类函数缺少鲜明的特征，且相关信息难以读取。然而，读者依然可直接获取渐进值。然而，大多数貌似拥有渐进值的函数仅是变化缓慢而已，例如图 3.5 中的 $y=\log_e(x)$函数，该对数函数并不包含最大值。

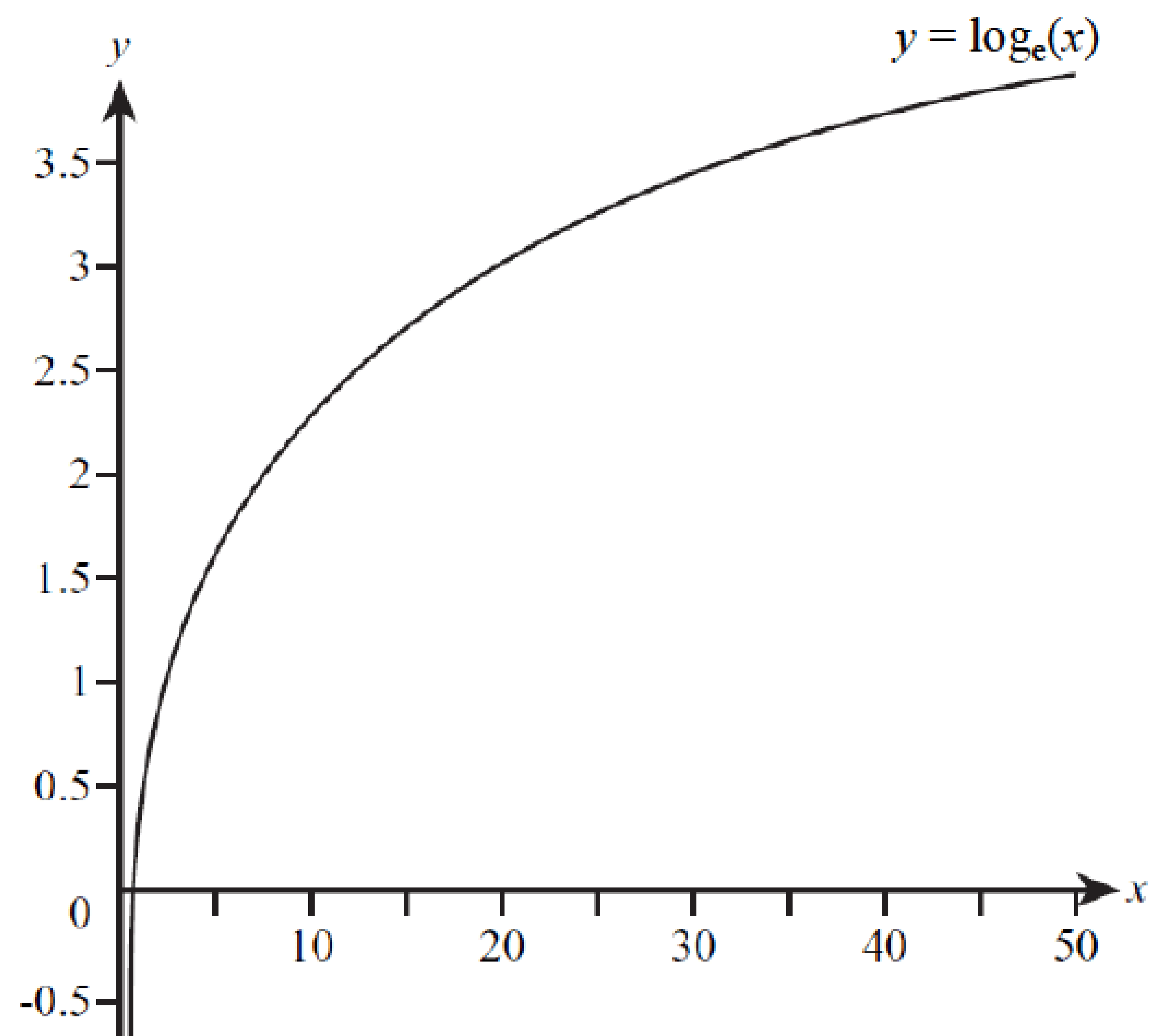


图 3.5　$y = \log_e(x)$函数图

3.5.4　参数曲线和函数

虽然简单函数可实现图形的绘制结果，但并非全部曲线均可通过标准形式予以描述。截止到目前为止，读者所考察的图形仅为单值函数，该技术无法绘制圆形图案——针对各 x 坐标，圆形并未包含单值 y，根据函数定义，该函数为多值函数。

参数化形式可避免多值函数产生的问题，也就是说，不使用单值函数 $f(x)$ 并标定 $y = f(x)$，读者可采用两个函数 $x(t)$和 $y(t)$，并针对各 t 值标定点$(x(t), y(t))$。此处，t 表示为虚变量。由于参数函数常用于表达运动，且后者常涉及时间值，因而 t 多表示为时间。

为了展示参数方程的工作方式，假设两个函数分别定义为 $x=at^2$ 和 $y=2at$，若 t 在实数范围内变化且绘制点(x,y)，则最终结果如图 3.6 所示。当替换参数公式中的 t 后，最终抛物线表示为 $y^2=4ax$。

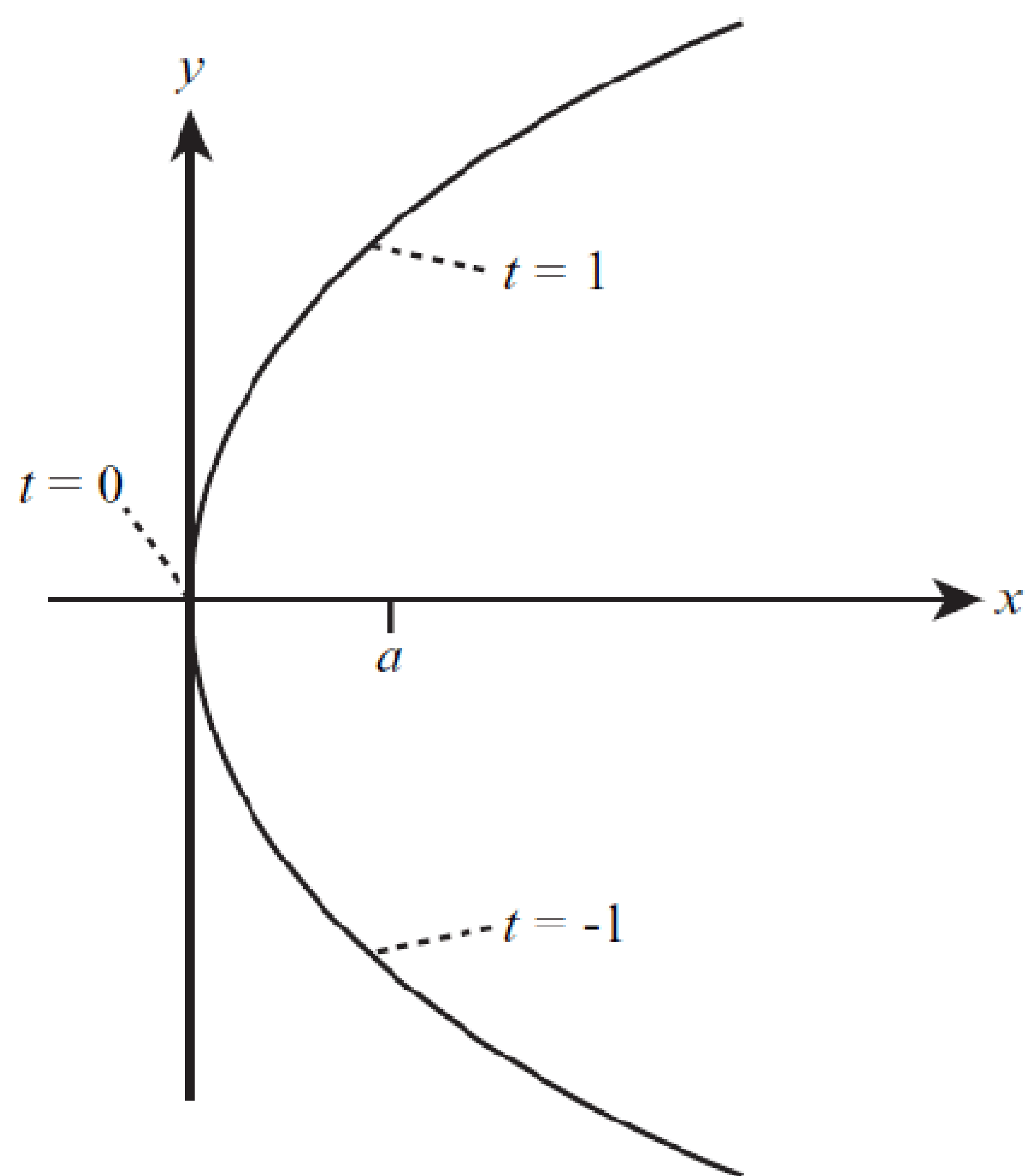


图 3.6　源自参数公式 $x = at^2, y = 2at$ 的抛物线

drawGraph()函数并不能完成该曲线的绘制工作，其原因在于，该函数为 y 坐标中的多值函数。稍后将会看到，与简单函数相比，参数公式可绘制更为复杂的曲线，包括 Bezier 曲线，以及向量绘制和 3D 建模软件包中的样条。

3.6 本章练习

【练习 3.1】试编写 substitute(functionString, x)函数，并将 x 值代入至标准符号形式的函数中。

该函数接收两个参数，例如形如“5x^2+3(4−2x)”的字符串以及字符串形式的变量 x。这里使用了前述章节所介绍的“^”符号表示指数，“/”和“*”则分别表示除法和乘法运算。该函数应具备一定的通用性并可处理括号问题。读者可对 5x+3，4−2(x−5)，(x^3−4)/2，(x−4)(2−3x)，2^((x−4)/(x−5))进行测试。

【练习 3.2】试编写 simplify(functionString)函数，并对既定函数实施简化操作。

简化工作需要一定的技巧，函数的工作方式无法与人类的计算方式媲美，但也应具备一定的功能。例如，程序应可接收某一函数、合并同类项以及基于公分母的分数计算，读者甚至还可进一步尝试因式分解操作。另外，函数应可与单变量或多变量协同工作。

【练习 3.3】试编写 solve(equationString)函数并对给定的方程进行求解。

如前所述，此类函数无法解决所有问题，但至少应可处理线性或二次方程。同时，读者还可借助于前述函数简化方案编写当前函数。

3.7 本章小结

本章讨论了基本的代数运算，进而帮助读者回顾某些细节内容以及相关练习。另外，编程技术的引入使得读者可更好地理解代数知识中的各种概念。

本章以相对高级的方式探讨了相关方法和概念，并将函数理念置于实际应用中。一旦读者理解了函数的真正含义，代数问题也将随之迎刃而解。第 3 章将稍稍放慢学习速度，并对几何形状加以考察，这也是与程序设计相关的另一个话题。

至此，读者应掌握如下内容：

- 理解术语变量、参数、处理以及未知项的含义，及其关联方式和彼此间的差别。
- 理解函数的具体含义，即不同数据集之间的映射，以及一对一映射、一对多映射和多值函数等概念。
- 方程的含义及其针对特定未知项的求解方式，包括二次方程、三次方程和基于两个或多个未知项的联立方程。
- 简化函数以及函数的因式分解运算，这对方程的求解很有帮助。
- 函数图的绘制方式，以及如何利用函数图读取对应函数的信息。
- 如何绘制参数曲线。

第 4 章　几何学和三角学

本章包含如下内容：

- 概述。
- 角度。
- 三角形。
- 三角形计算。
- 旋转和反射。

4.1　概　　述

几何学主要研究具有对称性的形状和空间，本章将集中讨论此类对象的实际计算过程，并首先介绍角度和三角形的计算，即三角学。当对运动行为进行编程时，常会使用到三角学，例如游戏。相关内容较为重要，这里也希望读者对此予以深入理解。

4.2　角　　度

角度可视为一种方向测量方法。如果两个人从同一点出发，待行进了 10 米后，二者间可相距 0 米（同一方向运动）或 20 米（相反方向运动），而角度则用于计算两个方向之间的差别。

4.2.1　角度和角度值

一种较为常见的角度测量方法是以圆半径的方式计算两个方向，并可将两个半径之间的角度视为一个分数。在图 4.1 中，直线 A 和 B 之间的角度占据了圆的 1/4，而直线 B 和 C 之间的角度值占据了圆的 1/3。

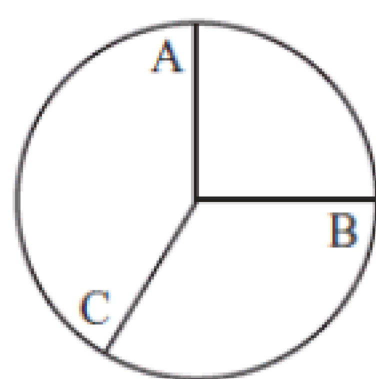


图 4.1　基于不同半径的圆

【提示】半径表示为圆心至周长之间的一条直线，其中，周长的也称作圆周。若圆周上两点之间绘制一条直线并穿越圆心，则该直线称作直径。

由于可在两种方向上测量角度，因而“两条直线之间的角度值”这一描述并不准确。在图 4.1 中，A 和 B 之间的逆时针角度占据了 1/4 个圆。一般而言，当介绍两直线之间的夹角时，通常意味着较小的角度，如无特殊说明，本章将采取这一方案。后续章节还将讨论特定方向上的角度测量方法。

角度可采用不同的单位进行计算，一种较为常见的方式是度数。当以度数方式测量角度时，可将圆形划分为 360 等份，各等份表示为 1°。此处，除了包含较多的除数之外，数字 360 并无特别含义，这也意味着，圆形对应的分数均包含整型度数。例如，1/4 旋转表示为 90°（称作直角），1/2 旋转表示为 180°（一条直线），1/3 旋转表示为 120°，1/6 旋转表示为 60°，1/5 旋转表示为 72°等，如图 4.2 所示。

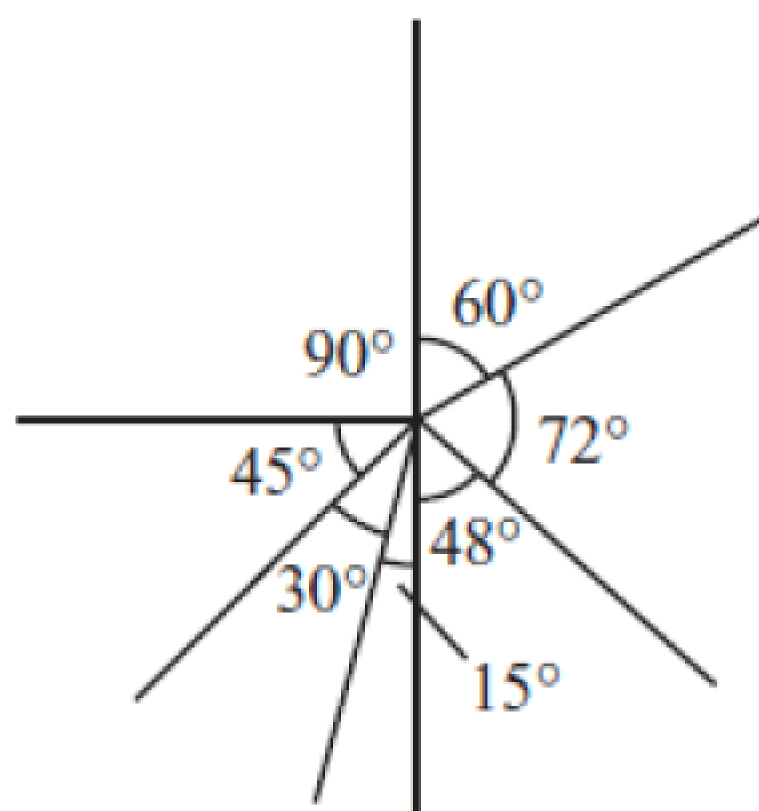


图 4.2　基于度数的角度测量。对应角度采用直线间的弧标记，而直角则采用部分正方形表示

在如图 4.2 所示的角度中，较为重要的是直角。正方形包含 4 条等边，各邻接边之间的夹角为直角。若绘制正方形的两条对角线，则二者交于直角。若两条直线相交后形成了直角，则直线间彼此垂直。另外，4 个直角将圆划分为 4 个均等象限。如图 4.3 所示，圆形形成的象限具有互补特征，针对圆内的(x, y)点，对应点为$(x, -y)$，$(-x, y)$和$(-x, -y)$。若 x 或 y 不为 0，此类点均位于圆内，且各自占据不同的象限。

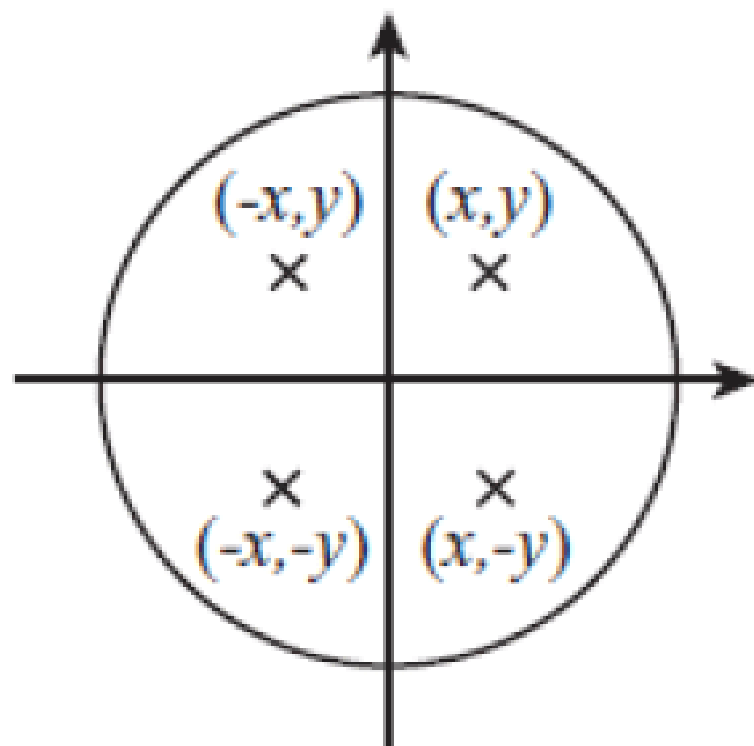


图 4.3　象限

某些术语则简化了角度相对值之间的讨论，例如，小于直角的夹角称作锐角，大于直角的夹角称作钝角，而大于半圆的夹角则称作反角（reflex angle）。

4.2.2　面积和 π

若围绕圆绘制一个正方形并在其中随机选取一点，则该点是否位于圆内？该问题涉及某一形状的面积计算。假设存在两个正方形，且对应边呈 2 倍关系。这里，将两个正方形并排置于桌面上，并在桌面上采用均匀间隔的点进行重复标记，则多少个点位于相应的正方形中？通过实践可知，位于较大正方形中的点数量 4 倍于较小正方形。其原因在于，4 个小型正方形填充于一个较大正方形中，因而 4 倍数量的点将落于较大正方形中。对此，单位正方形可用于扩展这一关系。若某一正方形的边长为 x 单元，则该正方形的面积为 x^2 平方单元。若 $x=1$，则该形状称作单位正方形。

据此，可计算出长方形的面积。类似于正方形，矩形包含 4 条边，且各邻接边交于直角。与正方形的 4 条边不同，矩形的 4 条边无须相等。在矩形中，仅对边之间保持相等。这也意味着，正方形可视为矩形的特例。矩形的面积表示为长度和宽度的乘积。在图 4.4 中，矩形的面积为 12 个平方单元，换而言之，12 个单位正方形可填满该矩形。

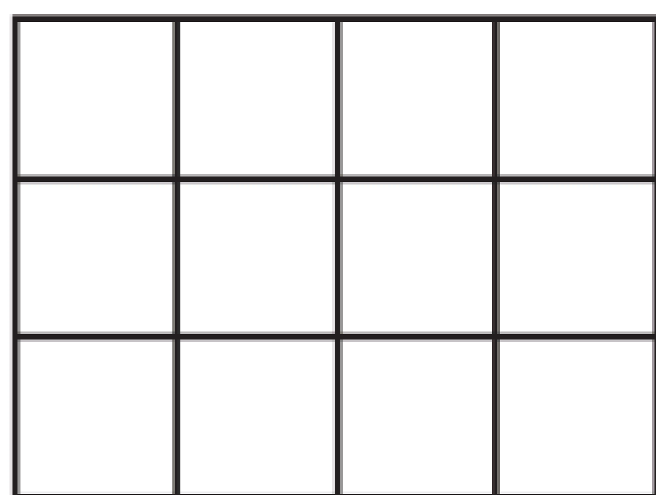


图 4.4　3×4 矩形

三角形依然会涉及面积问题，但此处首先讨论本节开始处提出的问题，即何为圆面积？在绘制圆形时，可将绳索一端置于某一固定点处，另一端则绑定一支笔，进而绘制圆弧。其中，绳索的长度即为圆半径。

【提示】术语“半径”是指长度和直线。相应地，圆半径表示为圆心至圆周之间的直线，同时也是该直线的长度。半径的使用方式类似于边，例如，正方形的边同时也意味着边长。同样，矩形边也指矩形一组对边的长度。因此，半径的含义也包含其长度值。类似地，直径也表示直径的长度值。

一旦了解了圆半径，则与圆相关的大部分问题亦迎刃而解，尽管半径仅需要知晓圆心的位置。圆形的面积与其半径形成的正方形之间存在特定的比例，若圆的半径为 r，其面积表示为 $3.1415927\cdots\times r^2$。其中，常量 3.1415927…的精确值采用符号 π 表示。该希腊字母 p 在英语中拼写为 pi，读作 pie。在大多数计算机语言中，该值可作为一个数学属性予以访问，并可预定义为 PI 或函数 pi()（后者较少出现）。

数学领域中经常可以看到 π 的身影，其频繁程度甚至超出了 e。几个世纪以来，科学家们针对该数字列举出了大量的事实，其中，数学家莱布尼茨定义了下列算式：

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

上式与莱布尼茨级数有几分类似。最早的计算机应用之一即是精确计算 π 的位数，多个级数项逼近于某一精确值。当前，π 的位数已超出了万亿个小数位，这也显示了数字的重要性以及计算机程序员对事物本质的钻研精神。

一旦理解了圆面积公式，即可回答与圆定义相关的问题。围绕圆所绘制的正方形，其边等于圆的直径，若圆半径为 1 个单位，则正方形边长可表示为 2 个单位，因此其面积为 4 单元^2。根据圆面积公式，圆面积可定义为 π 单元^2。因此，圆 π/4 倍于正方形面积，该值约为 0.7854。

π 还可视为圆形中的另一个重要元素，即圆周长。圆周长等于 π 乘以直径，或 π 乘以 2 倍的半径长。

4.2.3　弧度

如前所述，选取度数作为角度单位并无特殊原因，仅是出于方便计算考虑。除此之外，另一种常见的角度测算方式是弧度，尽管该方法初看之下并不自然。这里，圆不再划分为 360 个等份，而是分为 2π 个弧度——非整数单位并非是错误，这同英寸与厘米之间的换算关系出自同一个道理。

因此，角度与弧度之间的换算不可或缺。一个弧度可表示为圆的 $\frac{1}{2\pi}$，即 $\frac{360}{2\pi}$ 个角度。类似地，1° 表示为圆的 $\frac{1}{360}$，即 $\frac{2\pi}{360}$ 个弧度。据此可知，12° 等于 $\frac{12\times2\pi}{360}=\frac{\pi}{15}$=0.209 个弧度，90° 等于 $\frac{\pi}{2}$ 个弧度。若读者经常在两种角度值之间进行转换，则可将转换因子作为常量加以存储。

第 3 种角度单位则是梯度角（gradian），尽管某些计算器中包含了该值，但在实际应用过程中，梯度角并不常见。当采用梯度角时，圆将被划分为 400 份。

4.3　三　角　形

前述章节曾有所提及，作为数学中的一个重要领域，三角学主要研究三角形。同时，三角形也作为几何学中的一项课题被广泛研究。三角形可视为非共线 3 点构成的图形，并通过 3 条直线段予以连接。当数据点根据此方式定位时，则称作非共线状态。三角形中的各数据点定义为顶点，并构成了当前三角形的全部顶点。在向三角形和几何学中引入了代数后，该领域称作解析几何学并由 Rene Descarte 开创。

4.3.1　三角形类型

根据相应的角度，三角形可划分为 4 种主要类型，如图 4.5 所示。其中，三角形顶点采用大写字母表示，边长采用小写字母表示，角度则采用希腊字母或指定符号表示（例如∠ABC 显示了定义该角度的全部 3 个顶点）。类似地，各边还可根据相对的顶点进行标记；角也可通过顶点

字母加以表示，例如角 A，角 B 或角 C。另外，较短的弧可用于表示连接顶点的直线间的夹角。

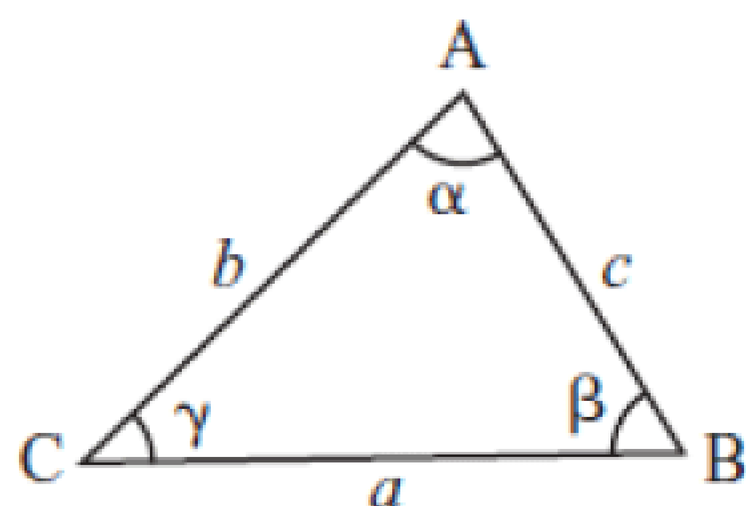


图 4.5　包含顶点、边和角度的三角形

【提示】如果读者对希腊字母感到陌生，可参考附录 C 以获取更多信息。作为快速浏览，α 表示 alpha，β 表示 beta，γ 表示 gamma，δ 表示 delta，ε 表示 epsilon，θ 表示 theta，而 π 表示 pi。

三角形的角度之和为 180°，在图 4.6 中，同一三角形被重复绘制 3 次。通过观察可知，角度 α，β，γ 位于同一直线上，这意味着，此类角度之和为 1/2 圆，即 180°。该结论由欧几里德（324～264 B.C.E.）提出，且适用于任意三角形。在几何学的雏形时期，欧几里德已名声大噪。

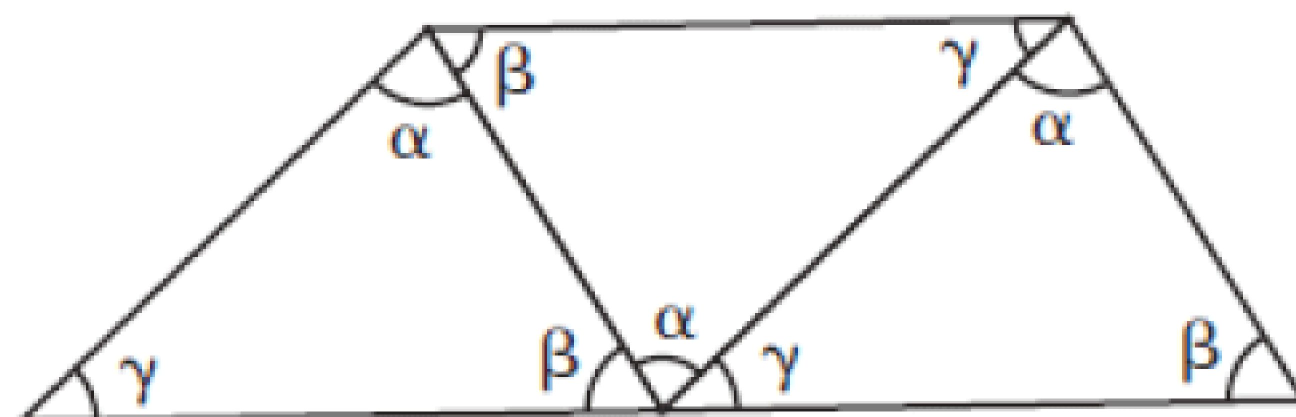


图 4.6　三角形内角之和等于 180°

4.3.2　通用三角形类型

尽管通用命题适用于全部三角形，且不考虑其对应形状，但三角形形状依然可根据角度值加以分类。总体而言，三角形可分为 3 类，如图 4.7 所示，下列内容对各类三角形进行了简要的总结：

- 等边三角形。等边三角形可视为最为简单的三角形，其边、角均相等。由于三角形内角和为 180°，因而等边三角形的各角均为 60°。
- 等腰三角形。等腰三角形包含两条等边，若 a 和 b 相等，则角 α 和 β 也相等。
- 不等边三角形。该三角形可为任意其他类型的三角形，且不包含等边和等角。

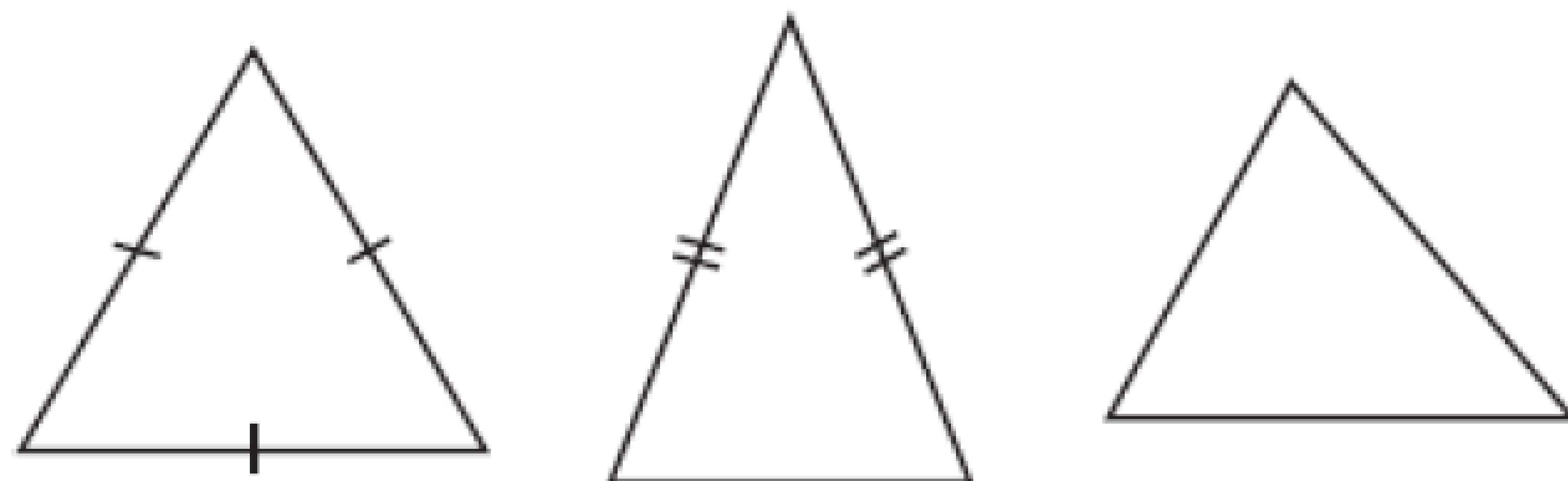


图 4.7　等边三角形、等腰三角形和不等边三角形

除了主要的 3 种三角形分类之外，人们还将三角形分为锐角三角形和钝角三角形。其中，钝角三角形中的一个角大于 90°；而锐角三角形中的各角皆不大于 90°。据此，等边三角形和等腰三角形可表示为锐角三角形，而不等边三角形则有可能为钝角三角形。

4.3.3 直角三角形

直角可视为一类较为重要的三角形类型，顾名思义，此类三角形中包含直角。其中，直角为 90°角，且直角三角形的各边具有特定的名称，即两条直角边和一条斜边。如图 4.8 所示，两条直角边（a 和 b）由直角连接。图中，直角由较小的正方形或矩形表示，而两条直角边的尾端由斜边连接。由于直角等于 90°，且三角形内角和为 180°，因而两个较小角度之和为 90°。

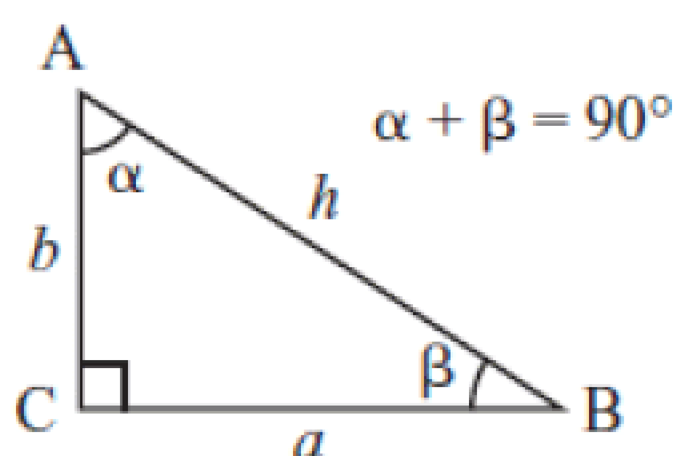


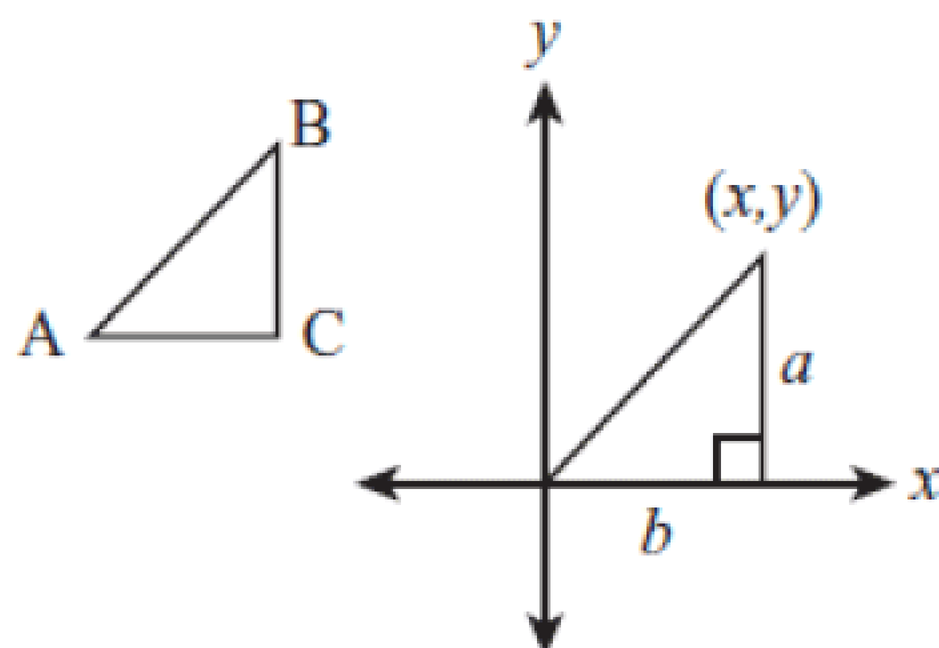
图 4.8 直角三角形

4.3.4 毕达哥拉斯定理

希腊数学家和哲学家毕达哥拉斯（569～494 B.C.E.）发现了直角三角形各边之间的关系，即毕达哥拉斯定理。笛卡儿和其他数学家对该定理进行了扩展，并将其应用于坐标系和代数运算中。

毕达哥拉斯定理的工作方式源自直角三角形，此类三角形的重要性主要体现于以下几点：首先，直角三角形易于识别，且适用于多种场合。例如，当与函数图协同工作时，常可绘制连接多个点的直线，并通过平行于 x 或 y 轴的各边生成直角三角形，如图 4.9 所示。如前所述，直线的梯度或斜率由两边（a 和 b）或 x、y 轴上的 y/x 确定。

三角形：梯度（或斜率）$=a/b$



坐标系：梯度（或斜率）$=y/x$

图 4.9 直角三角形

其次，直角三角形包含许多有用的属性，稍后将对此加以分析。最后，任意三角形均可划分为两个直角三角形，即通过某一顶点向对边绘制一条垂直边，如图 4.10 所示。

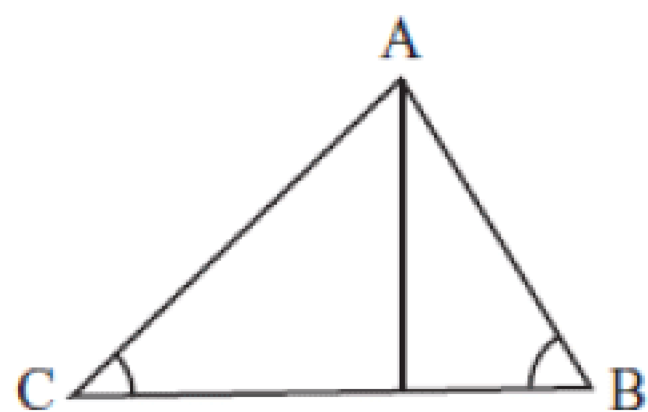


图 4.10　从顶点 A 绘制一条垂直线，三角形可划分为两个直角三角形

毕达哥拉斯对直角三角形进行了深入研究，研究结果表明，在直角三角形 ABC 处（直角位于顶点 C），有 $a^2+b^2=c^2$。对此，存在多种方法可对其加以证明，图 4.11 显示了其中的一种方案。

在图 4.11（a）中，正方形的边长为 $a+b$。其中，外部 4 个直角三角形的直角边分别为 a 和 b。三角形的斜边 c 构成了内部正方形，其面积为 c^2。

图 4.11（b）显示了同样的正方形，其边长为 $a+b$。此时，4 个直角三角形经重组后填充于该正方形中，图中显示为两个矩形交于一点。其中，各矩形的边长分别为 a 和 b。当前正方形其余部分被划分为两个较小的正方形，边长分别为 a 和 b。

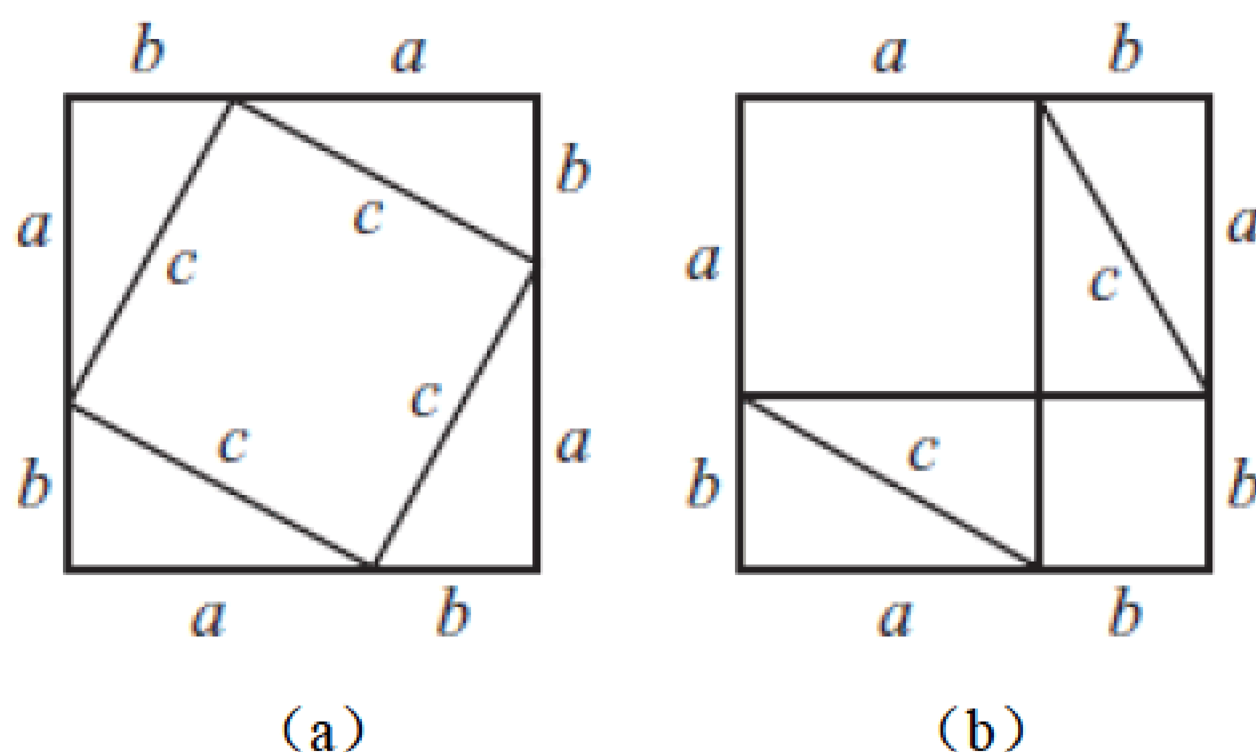


图 4.11　毕达哥拉斯定理的几何证明

两个较小正方形的全部面积为 a^2+b^2，而两个较大正方形以及其中的 8 个直角三角形全部相同，因而图 4.11（b）中两个较小正方形的面积等于图 4.11（a）中较小正方形的面积。通过上述观察可知，$a^2+b^2=c^2$。

4.3.5　毕达哥拉斯三元数

不难发现，存在无数多个直角三角形，其边长为整型长度值。其中，最小直角三角形的直角边分别为 3 和 4，且斜边为 5。对此，存在多种方式可生成此类三角形，且构成各边的 3 个数字集称作毕达哥拉斯三元数。而对于较大阶数，情况则有所变化。也就是说，针对 $a^n+b^n=c^n$（$n>2$），不存在正整数集合 a，b，c，n，即费马大定理，该问题直到近期方得以解决。

4.3.6　毕达哥拉斯定理推论

当使用毕达哥拉斯定理时，若给定两条直角边的长度，则可快速确定第 3 条边的长度。例如，若已知斜边长度为 13 厘米，一条边长为 5 厘米，则根据毕达哥拉斯定理，第 3 条边长为

$\sqrt{13^2-5^2}=\sqrt{144}$ =12 厘米。

在图 4.12 中，等腰直角三角形表示为正方形的一半，且直角边为 1（通常称作单位长度），则斜边为$\sqrt{2}$ =1.414…个单位长。类似地，若等边三角形边长为 2 并将其一分为二，则垂线的长度为$\sqrt{3}$个单位。需要注意的是，等腰三角形的内角为 45°，而包含垂线（值为$\sqrt{3}$）的三角形其内角为 60°和 30°。

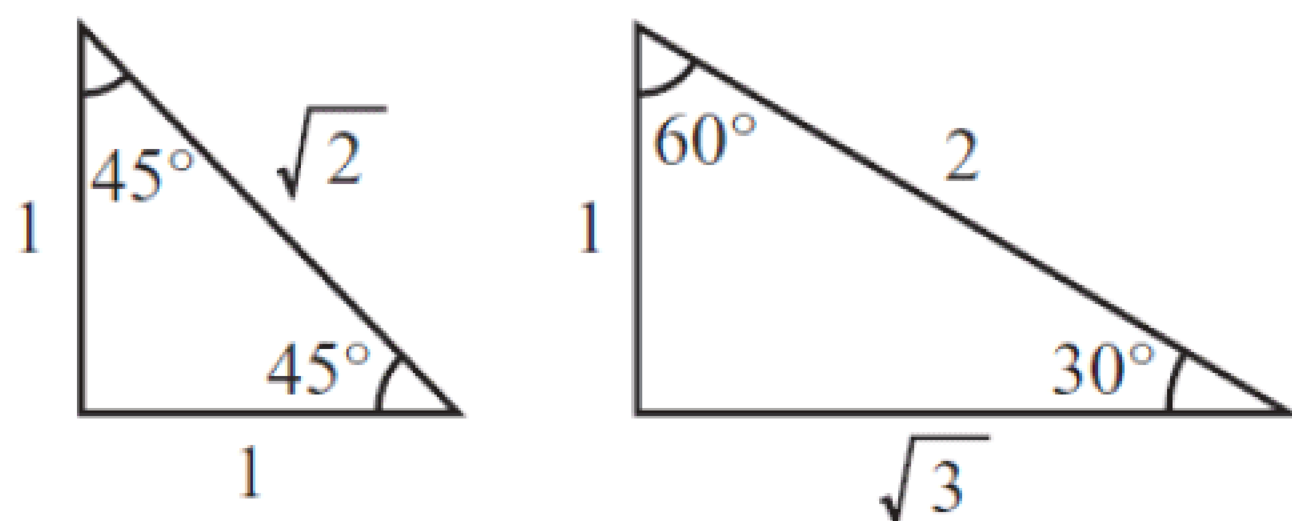


图 4.12　较为重要的直角三角形

4.3.7　三角函数

回顾图 4.9，其中，直线的梯度（或斜率）可通过直角三角形表达。针对任意直角三角形（直角边 a 平行于 y 轴，直角边 b 平行于 x 轴），则斜边的梯度为$\frac{a}{b}$。通过 tan()函数，可将该值与 A 和 B 处的角度进行关联。若 A 处的角度为 x，则 tan(x)将生成直线 AB 的梯度。sin(x)和 cos(x)则分别等于$\frac{a}{c}$和$\frac{b}{c}$。上述 3 个函数称作三角函数，且有 tan(x)=sin(x)/cos(x)。

【提示】 tan 表示为正切函数的缩写，余弦函数的缩写为 cos，正弦函数的缩写则表示为 sin。需要说明的是，三角函数取决于角度的测量单位。总体而言，计算机语言假设采用弧度单位，而出于简洁考量，本小节依然采用度数单位。因此，当使用度数单位时，应将其转换为弧度单位以供计算机程序设计语言使用。对此，可将角度值乘以 π/180。

在图 4.12 中，通过观察可知，sin(45)等于 1/sqrt(2)，sin(30)等于 0.5，sin(60)等于 sqrt(3)/2。而 cos 值和 tan 值也可具有类似的计算方式。

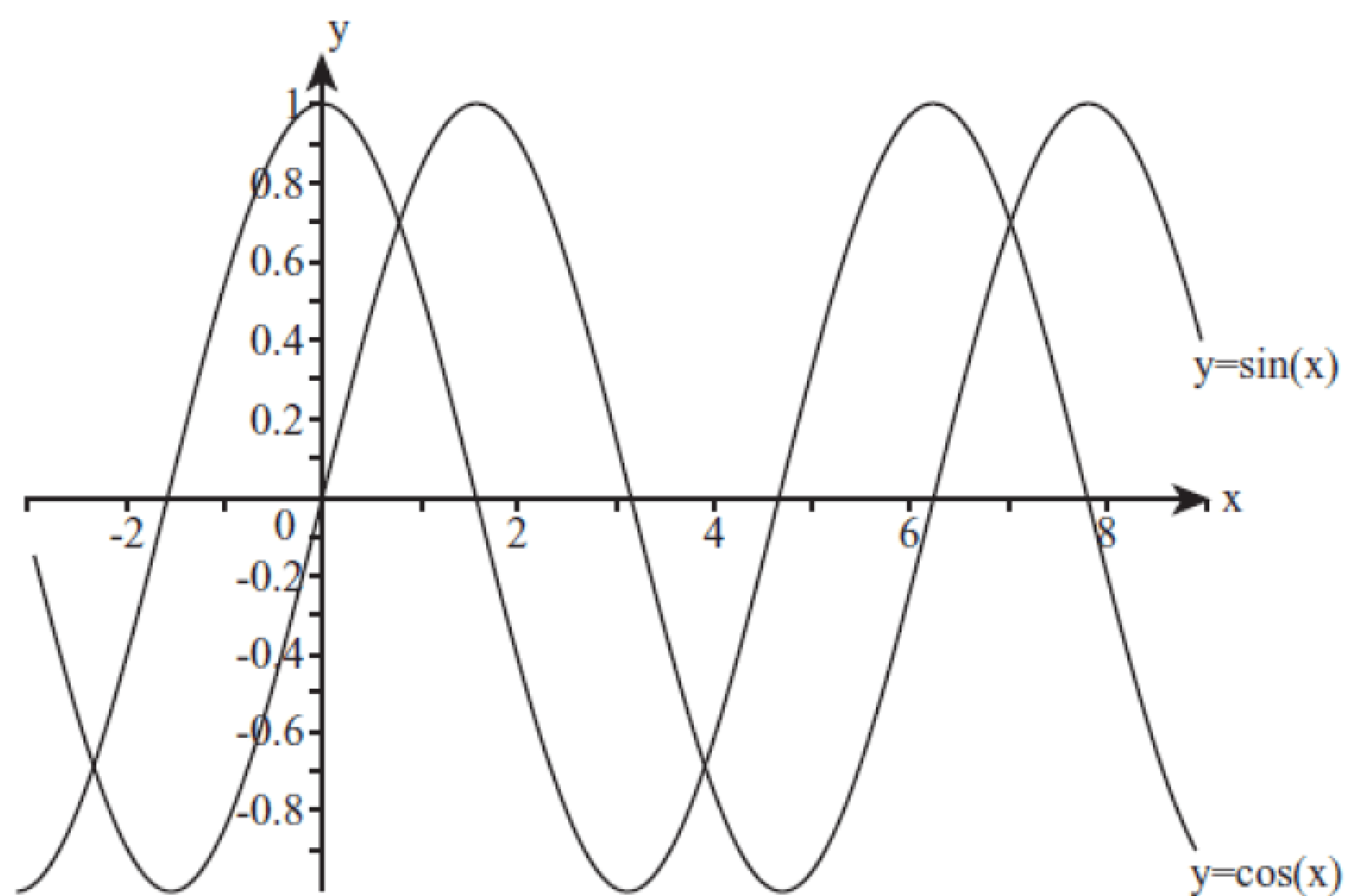
若针对各直角三角形计算此类函数值，并绘制函数图，对应结果十分有趣，如图 4.13 所示。图 4.13（a）显示了 0°～360°之间的 $y=\sin(x)$和 $y=\cos(x)$的函数图。需要注意的是，类似于梯度的测量方法，读者可在两个方向上计算边长。不难发现，对应函数图具有相同的形状，即连续的正弦波形。其中 cos(x)波形稍落后于 sin(x)，该现象称作 90°反相。

图 4.13（b）显示了 $y=\tan(x)$的函数图，且与正弦波形明显不同，其中一个显著特征是，不再生成连续曲线，相反，该函数每隔 180°生成一系列的曲线。

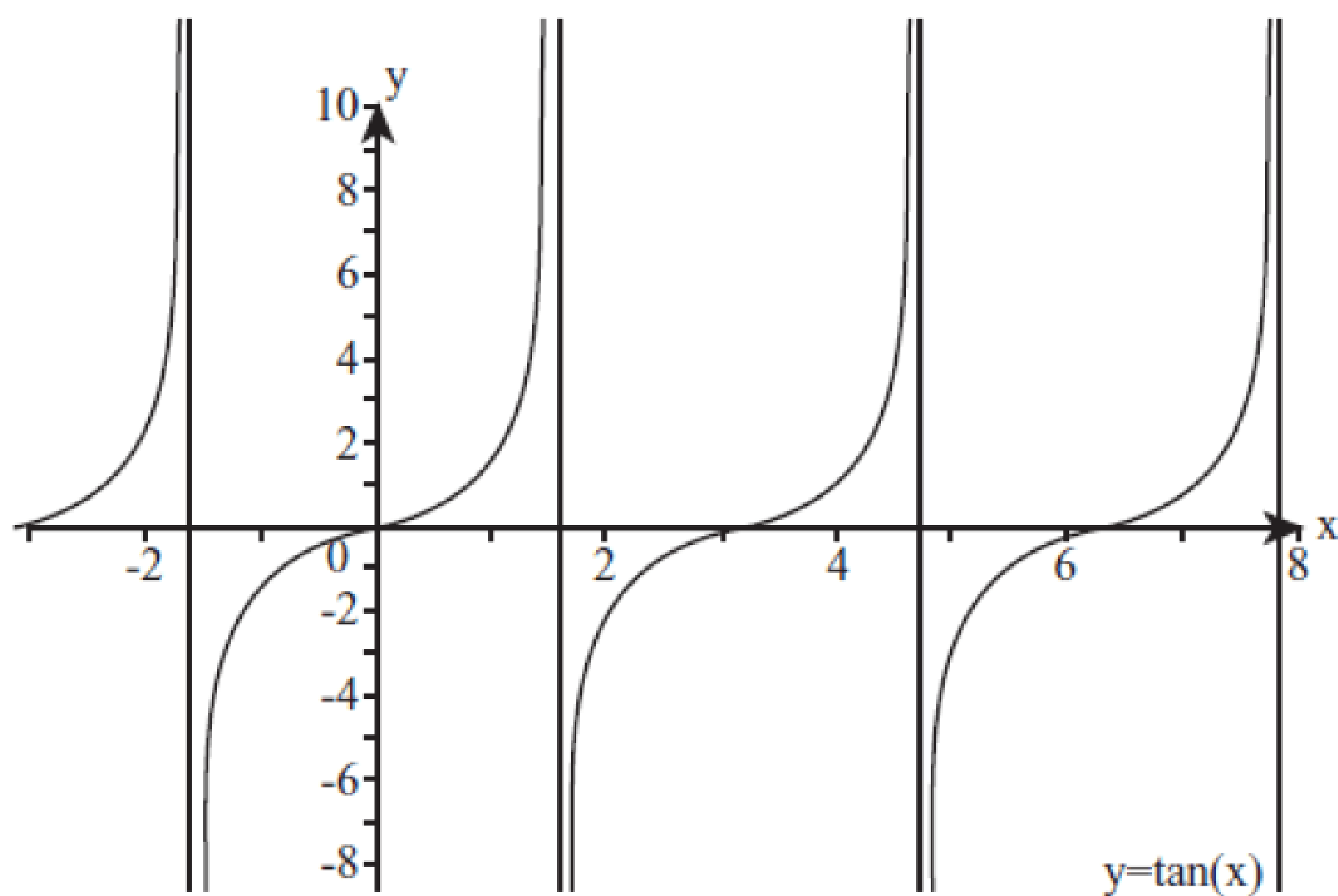
函数 sin()和 cos()与圆周运动关联紧密，第 16 章将对此加以深入讨论。类似地，此类函数的行为与 exp()函数也较为接近。对应函数均可通过下列无穷级数进行计算（采用弧度制）：

$$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\cdots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$



（a）



（b）

图 4.13　sin(x)、cos(x)和 tan(x)的函数图

需要注意的是，当 x 值较小时，sin(x)几近等于 x。除此之外，sin(0) = 0，cos(0)=1。

4.3.8　三角恒等式

为了对三角函数进行有效的扩展，因而衍生出了多个三角恒等式。对应等式均未经证明，并

留与读者以作练习。针对任意 x 和 y，有：

- $\sin^2(x)+\cos^2(x)=1$。其中，$\sin^2(x)$为$(\sin(x))$的简写形式。
- $\sin(x+y)=\sin(x)\cos(y)+\cos(x)\sin(y)$。
- $\cos(x+y)=\cos(x)\cos(y)-\sin(x)\sin(y)$。
- $\tan(x+y)=\dfrac{\tan(x)+\tan(y)}{1-\tan(x)\tan(y)}$。
- $\sin(2x)=2\sin(x)\cos(x)$。
- $\cos(2x)=\cos^2(x)-\sin^2(x)$。
- $\tan(2x)=\dfrac{2\tan(x)}{1-\tan^2(x)}$

4.3.9 反三角函数

根据角度计算梯度十分重要，而根据梯度计算角度值同样不可或缺。上述 3 个函数均包含反函数，即 asin、acos 和 atan，分别对应于 arcsine 、arctangent 和 arccosine 的缩写。各个反函数在定义域和值域范围内进行映射，且 3 个函数的数据域均为[–1,1]。相应地，asin 和 acos 映射至[–∞,∞]内，而 tan 则映射至[0,360]中。各函数均接收一个参数，并将其映射为多个角度值，因而此类函数定义为多值函数。针对各函数，存在一个标准的映射过程，如下所示：

- sin()的反函数定义为 asin()，或者表示为 arcsin()和 $\sin^{-1}()$。该函数将位于 0～1 之间的正值映射为 0～90° 之间的数据值，而大于等于–1 的负值映射为 90° ～180° 的数值。针对全部 x 值，有 $\arcsin(-x)=180-\arcsin(x)$。
- cos()的反函数定义为 acos()，或者表示为 arccos()和 $\cos^{-1}()$。该函数将[0,1]范围内的数值映射为[–90° ,0]，并将[–1, 0]范围内的数据值映射至[–90° ,0]。针对全部 x 值，有 $\arccos(-x)=-\arccos(x)$。
- tan()的反函数定义为 ztan()，或者表示为 $\tan^{-1}()$和 arctan()。该函数将[0,∞]中的数值映射至[0,90°)范围内。在定义域[–∞,0]中，函数将映射至(–90° ,0]。针对全部 x 值，有 $\arctan(-x)=-\arctan(x)$。

【提示】这里，定义域和值域均表示为区间形式，其中方括号和圆括号分别表示“包含”和“不包含”。例如，$[a,\ b]$表示为 a～b 的闭区间，即包含 a 和 b 的实数集；而$(a,\ b)$则表示 a 和 b 之间的开区间，即不包含 a 和 b 的实数集。类似地，$[a,\ b)$是指大于或等于 a 且小于 b 的数字集。

在数学领域中，反三角函数应用于多种场合。但对于程序设计而言，当与毕达哥拉斯定理结合使用时，arctan()函数依然可满足大多数需求。针对图 4.14，下列两个方程显示了比值（斜边与对边 x 之比）与 arctan()参数之间的应用方式，进而生成等于 sin 和 cos 反函数的数据值。

$$\cos^{-1}(x)=\tan^{-1}\left(\frac{\sqrt{1-x^2}}{x}\right)$$

$$\sin^{-1}(x) = \tan^{-1}\left(\frac{x}{\sqrt{1-x^2}}\right)$$

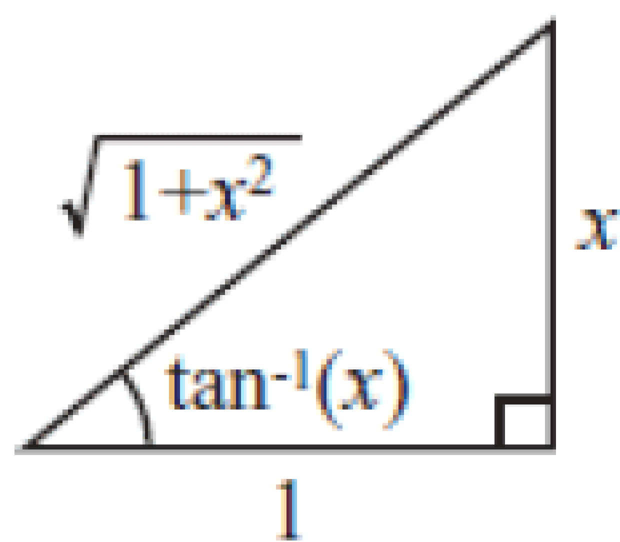


图 4.14　根据 arctan()函数计算 arcsin()和 arccos()函数

由于 arctan()还是将无穷大值映射至有限值，因而可处理分母为 0 的分数。当然，程序设计语言并不能直接使用此类函数。对此，可定义 arctan()函数的特定版本，以使其接收两个参数（而非一个参数）。其中，两个参数表示直角三角形的两条直边，且某一条边可能为 0。该函数在程序设计语言中常定义为 arctan2()或 atan2()，下列代码显示了其实现过程：

```
function atan2(y, x)
  set deg=1
  if x=0 and y<0 then deg=90
  if x=0 and y>=0 then deg=-90
  if y=0 and x<0 then deg=0
  if y=0 and x>=0 then deg=-180
  if deg=1 then return arctan(y/x)
  otherwise return deg*pi/180
end function
```

4.4　三角形计算

根据三角函数和毕达哥拉斯定理，读者可解决与三角形或复杂图形相关的问题，本小节讨论其数学内容以及计算方案。

4.4.1　正弦和余弦定理

当与三角形协同工作时，如何对其求解则成为了首要问题。该过程需要使用到与三角形相关的部分信息，进而获取与其相关的全部内容，包括角度和边长。例如，在图 4.15 中，若给定三角形 ABC 及其有限的边、角信息，根据三角函数和毕达哥拉斯定理，则可获得与 a，b，c，α，β，γ 相关的数值。当对三角形进行全方位求解时，读者应获得如下信息：

- 3 条边的长度。
- 任意两个角度和一条边的长度。

- 任意两条边的长度和二者的夹角。
- 在直角三角形中，任意两条边的长度。

除了直角三角形外，仅涉及两条边以及非夹角则无法求解当前三角形，其原因在于，可能存在两个三角形可满足这一条件，如图 4.15 所示。另外，若仅知 3 个角度值，则三角形依然无法进行求解——若边长成比例增长或降低，则对应角度值保持不变。

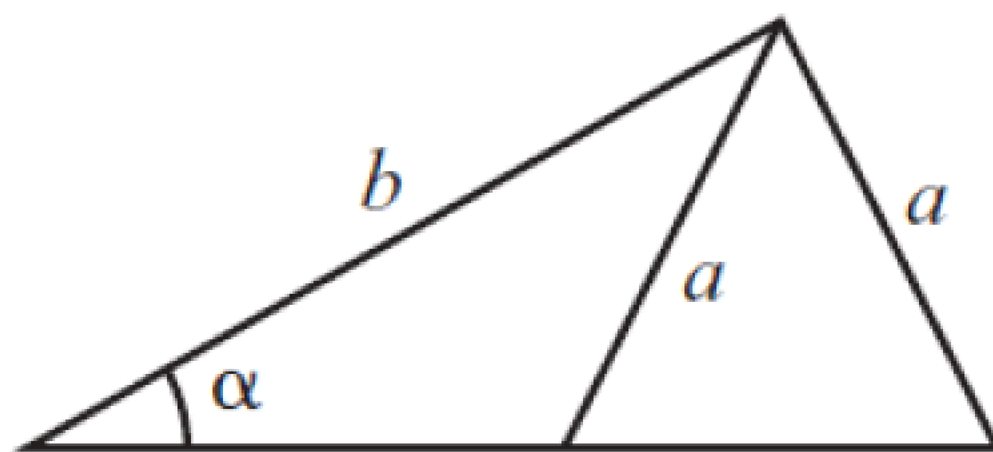


图 4.15　具有相同边值 a，b 和角度值 α 的两个三角形

正弦定理和余弦定理可视为求解三角形时的两个主要方法。其中，正弦定理将角度域对边关联，而余弦定理则将某一角度与 3 条边关联。

针对任意三角形，正弦定理如下所示：

$$\frac{a}{\sin(\alpha)}=\frac{b}{\sin(\beta)}=\frac{c}{\sin(\gamma)}$$

在图 4.16 中，经顶点 B 引入一条垂线，并与直线 AC 交于点 P，且 BP 的长度表示为 l。据此，可得到如下两个方程：

$$\sin(\alpha) = l/c \qquad ①$$

$$\sin(\gamma) = l/a \qquad ②$$

若重组方程②并将①代入至方程①中，则可得到下列方程：

$$\sin(\alpha) = \frac{a\sin(\gamma)}{c}$$

$$\frac{a}{\sin(\alpha)}=\frac{c}{\sin(\gamma)}$$

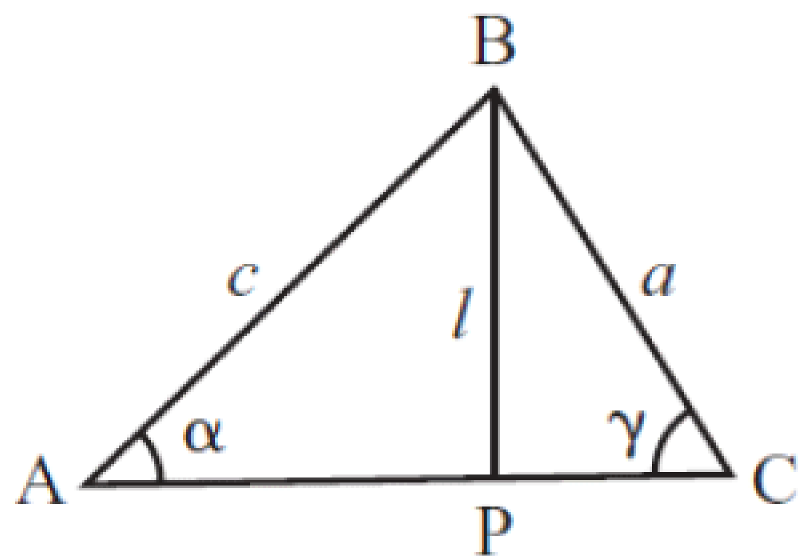


图 4.16　正弦和余弦定理的证明

同样，对称参数也可应用于 b 和 β 上。

余弦定理则稍显复杂，但作为毕达哥拉斯定理的推论，该定理易于记忆，如下所示：

$$a^2 = b^2 + c^2 - 2bc\cos(\alpha)$$

针对上式，需要再次考察图 4.16。其中，线段 PC 记为 k 个单位，而 AP 为 $b-k$ 个单位（全部线段为 b 个单位）。根据毕达哥拉斯定理，可得到如下等式：

$$a^2 = l^2+k^2 \text{（使用三角形 BCP）}$$

$$l^2 = c^2-(b-k)^2 \text{（使用三角形 ABP）}$$

消除 l 后，可得到下列等式：

$$\begin{aligned} a^2 &= c^2+k^2-(b-k)^2 \\ &= c^2+k^2-b^2+2bk-k^2 \\ &= c^2-b^2+2bk \end{aligned}$$

由于 $\cos(\alpha)=b-\frac{k}{c}$（通过三角形 ABP）且 $k=b-c\cos(\alpha)$，因而可通过 α 角消除 k 项，并得到如下等式：

$$\begin{aligned} a^2 &= c^2-b^2+2b(b-c\cos(\alpha)) \\ &= c^2-b^2+2b^2-2bc\cos(\alpha) \\ &= b^2+c^2-2bc\cos(\alpha) \end{aligned}$$

当结合使用正弦定理、余弦定理以及毕达哥拉斯定理时，可知三角形内角和为 180°，并可据此对三角形进行求解（参见练习 4.1）。

4.4.2　相似三角形

在前述章节中，两个三角形可拥有相同的角度值，但边长却彼此并不相等。具有相同角度的两个三角形称作相似三角形，由于三角形内角和为常量，因而仅需两角相等即可知三角形处于相似状态。

相似三角形原理表明，各边之间处于同一比例。正如第 2 章所讨论的矩形，从本质上讲，相似三角形之间均为同一三角形，只是按照不同比例进行绘制而已。这也意味着，若两个三角形之间呈相似关系，则可通过某一三角形中的边长推断出另一个三角形中的对应边尺寸。

图 4.17 显示了日光下测量建筑物高度时采用的方法。这里，可将 1 米长的木棍垂直矗立于地面上，并测量阴影的长度以及建筑物阴影的长度。其中，T、S、U 构成的三角形与 B、A、C 构成的三角形为相似三角形，S 和 A 处的角度皆为直角三角形，而 U 和 C 处的角度等于日光与地面之间的夹角。为了获取日光的精确角度，需要对齐木棍，以使 U 和 C 处于同一平面。

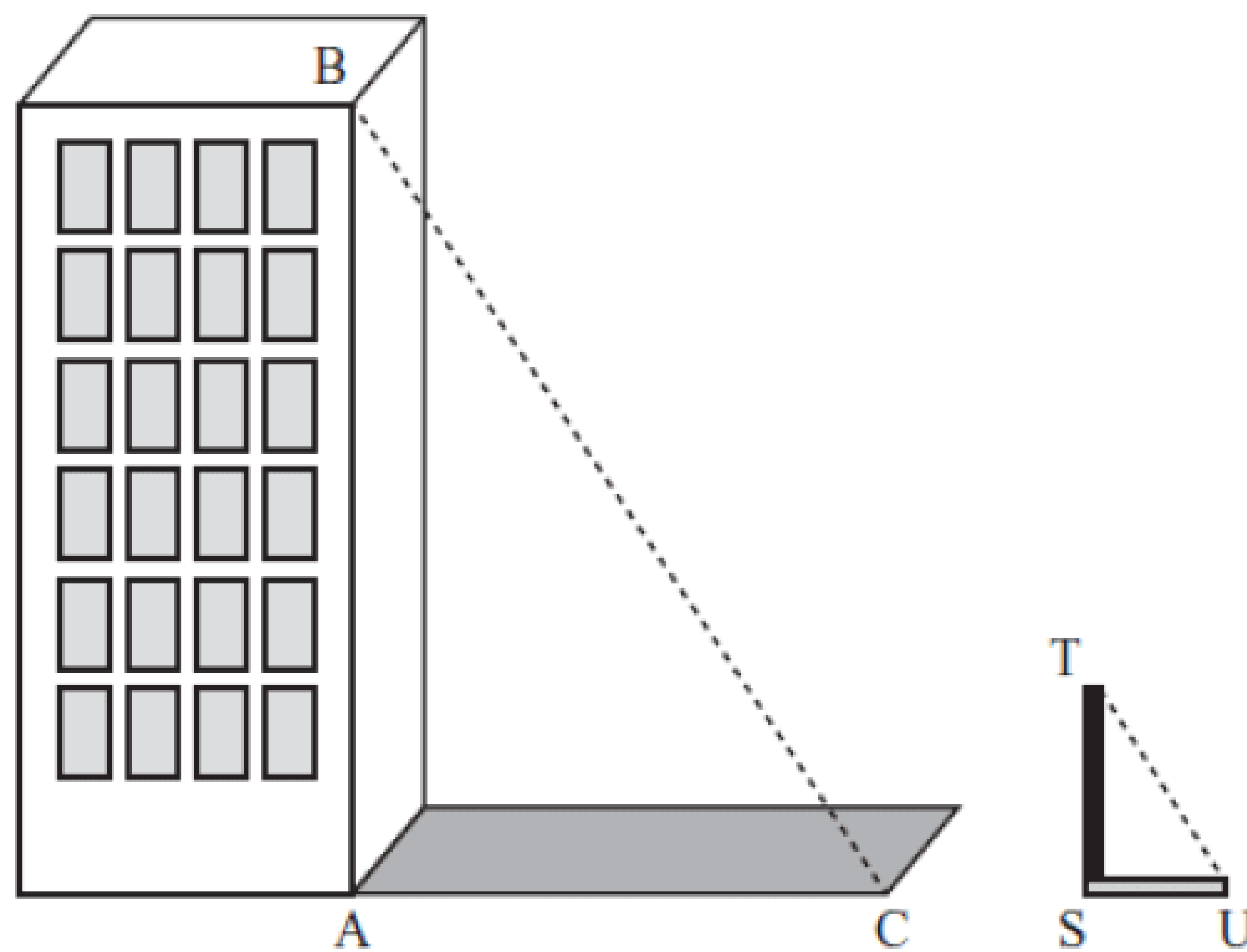


图 4.17　使用相似三角形测量建筑物的高度

【提示】当描述两个三角形之间的相似程度时，需要以相应的顺序对二者加以表示。换而言之，若三角形 ABC 和三角形 PQR 为相似三角形，则角 A 和角 P、角 B 和角 Q、角 C 和角 R 彼此相等。

由于图 4.17 中的三角形为相似三角形，因而其对应边呈相同比例。其结果为，建筑物 AB 的高度及木棍 ST 高度之间的比例等于建筑物阴影 AC 与木棍阴影 SU 之间的比例。通过代数形式，上述比例如下所示：

$$\frac{c}{u}=\frac{b}{t}$$

通过上式，可根据其他 3 个长度值计算 c，前者皆可从地面处进行测量。

【提示】A 与 B 之间的直线长度记为 AB，精简符号在复杂问题的表达中十分有用，并采用正体表示。当采用粗体或上方箭头形式表达时（例如 $\overline{AB}$），则对应符号为向量而非长度值，第 5 章将对向量加以深入讨论。

若两个三角形的边、角均相同，则二者称作全等三角形。也就是说，两个三角形彼此相同，或彼此为镜像图像。全等这一概念常出现于几何证明中，而较少出现于程序设计中。

4.4.3　三角形面积

有多种方案可计算三角形面积，当然，实际操作还取决于具体计算方法。例如，可通过如图 4.16 所示的三角形推导出一种计算方法，该方法使用三角形 ABC，并绘制顶点 B 与 AC 上的点 P 之间的垂线 l，这将原三角形划分为两个直角三角形。

随后，可在两个三角形旁绘制全等三角形，进而生成两个邻接矩形，且 2 倍于对应的直角三角形。该矩形的全部面积 2 倍于三角形 ABC 的面积，即三角形的面积可表示为$\frac{1}{2}$(底边×高)。

当然，还可通过其他计算方式获取面积值。例如，考察三角形 ACP，不难发现 $l=b\sin(\gamma)$，因而可将该值代入至前述公式中，进而得到面积值$\frac{1}{2}ab\sin(\gamma)$。

4.5　旋转和反射

当与旋转对象协同工作时，相关程序常会使用到三角函数。本小节将讨论三角形函数与旋转之间的关系。

4.5.1　转换

绘制于屏幕上的对象可通过顶点位置加以描述。在图 4.18 中，三角形（T）通过监视器屏幕

的不同位置加以表达。回忆一下，在前述章节中曾有所提及，y 轴的(0,0)坐标表示屏幕的左上角。相应地，初始状态下，三角形 T 包含顶点(100,150)，(125,125)和(150,130)，据此，存在多种方式可将三角形移动至新位置，即转换操作。

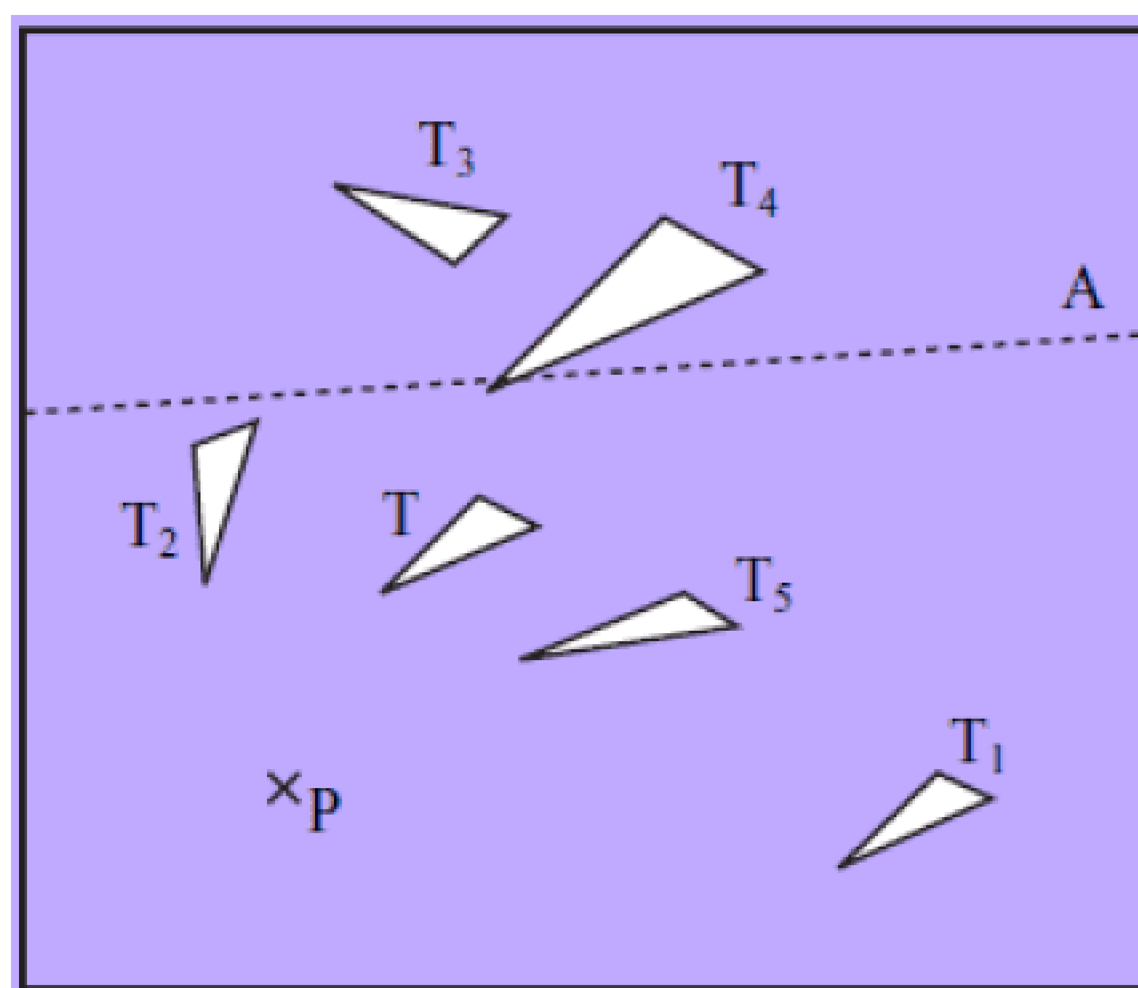


图 4.18　三角形及其转换方式

根据图 4.18，下列内容显示了三角形 T 的转换方式：

- 平移。若将某一角色移动至新位置且不包含旋转操作（例如三角形 T_1），这一变化称作平移。平移可通过向量加以描述，详细内容将在第 5 章加以讨论。当前，读者可将其视为一个值，进而确定该角色在 x 轴和 y 轴上的移动距离。
- 旋转。若将角色转动某一角度（例如三角形 T_2），该变化则称作旋转。这里，旋转行为通过一个点和角度进行描述。具体而言，三角形 T_2 可表示为“三角形 T 围绕点 P 旋转 45°”。其中，该点称作旋转中心，对应角度称作旋转角，且在图中采用顺时针方式测算，而在计算机中，旋转角多采用逆时针方式计算。
- 反射。若翻转图像（例如三角形 T_3），则该操作称作反射。反射通过一条反射轴进行定义。例如，三角形 T_3 可表示为“在轴 A（即直线 $y=x$）上反射三角形 T”。
- 缩放操作。若改变角色的尺寸（例如三角形 T_4），则该行为称作缩放操作。缩放操作可通过一点和一个数字值（即新、旧角色尺寸比例值）进行定义。其中，独立点表示为缩放对象的原点，且不应与图像的原点产生混淆；数字值则称作缩放因子并表示为一个比例值。当上述规范确定完毕后，T_4 可表示为“相对于点 P 并采用缩放因子 1.5 对三角形 T 执行缩放操作”。
- 剪切操作。若拾取三角形上的一点并相对于其他点移动，则称作剪切操作（例如三角形 T_5）。剪切操作可根据一条恒定直线和表示剪切量的一个数字（剪切因子）加以定义。

通过对各顶点执行各类操作，即可采用数值方式对上述转换进行计算，大多数计算均会涉及前述内容所讨论的三角函数。

4.5.2　旋转对象某一角度

如图 4.19 所示，三角形 T（即三角形 ABC）围绕原点 O 旋转 α 角，并形成新三角形 T'（即

三角形 A'B'C'）。对此，可使用监视器坐标系中的 y 轴，该轴指向屏幕下方。为了描述旋转操作，可从原点 O 至点与 A = (x,y)和 A' = (x',y')绘制直线。同时，从 A 和 A'向 x 轴和 y 轴绘制直线并分别与各轴交于 X，X'，Y，Y'点。

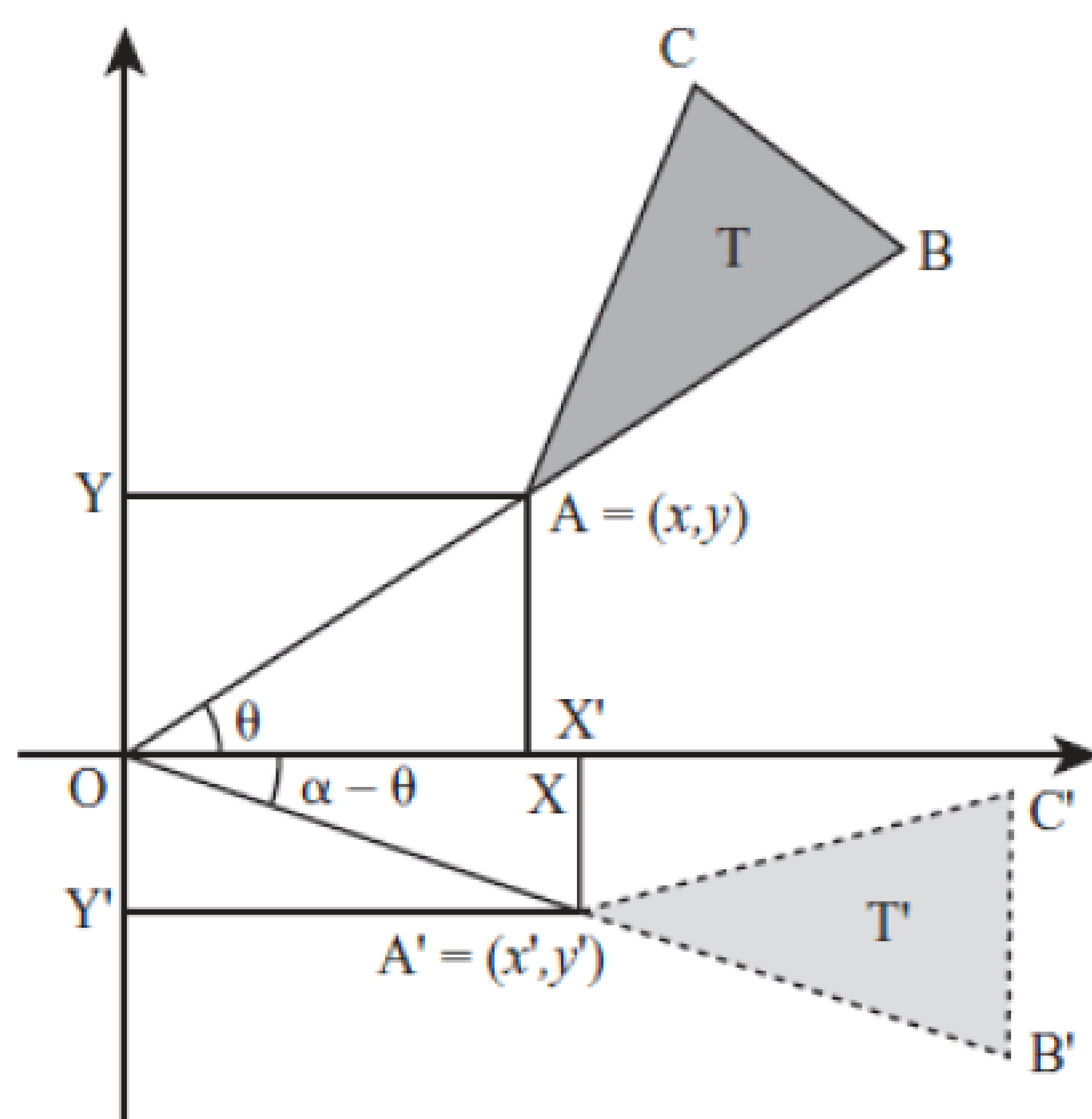


图 4.19　围绕原点进行旋转

除此之外，直线 OX 和 OY 的长度分别为 x 和 y，而直线 OX'和 OY'的长度为 x'和 y'。进一步讲，OA 和 OA'的长度保持相同（源自旋转的定义特征）。

从上述信息中读者可得到何种结论？首先，由于 OA = OA'，根据毕达哥拉斯定理，可知 OA' $=\sqrt{x^2+y^2}$；其次，可计算 OA 和 x 轴之间的角度值，即 atan(y, x)（图中的 θ）。

由于 α 表示为旋转角，则 OA'和 x 轴之间的夹角 θ' 等于 $\alpha-\theta=\alpha-\text{atan}(x,y)$。这也意味着，可按照下列方式计算 x'和 y'值：

$$x'=\text{OA}'\times\cos(\theta)=\sqrt{(x^2+y^2)}\cos(\alpha-\text{atan}(y,x))$$

$$y'=\text{OA}'\times\sin(\theta)=\sqrt{(x^2+y^2)}\sin(\alpha-\text{atan}(y,x))$$

针对各顶点重复上述操作即可使三角形整体围绕原点旋转。

【提示】当计算 x'和 y'时，应谨慎处理 x 和 y 的符号问题。对于正确的角度返回值，atan()函数应采用双参数实现方案，即 atan(y, x)。

若围绕任意一点而非原点旋转对象，情况又当如何？对此，首先需要移动对象并使旋转中心位于原点位置。随后，可旋转该对象并将其再次移回，该方案源自向量这一理念（第 5 章将对向量进行深入讨论）。

若围绕点 P=(s,t)旋转一点 α 角，则需要从各点中减去(s, t)进而对其进行平移。随后，可围绕原点旋转 α 角。最终，可向结果点数据中加入(s, t)，对于 x'和 y'值，这将生成较为复杂的公式，如下所示：

$$x'\sqrt{((x-s)^2+(y-t)^2)}\cos(\alpha-\text{atan}((y-t),(x-s)))+s$$

$$y'\sqrt{((x-s)^2+(y-t)^2)}\sin(\alpha-\text{atan}((y-t),(x-s)))+t$$

通过考察上述公式的关系可知，各公式分别采用 $x-s$ 和 $y-t$ 替换 x 和 y，并在适当时候添加 s

或 t。通过该方式对转换操作进行组合，则可生成强有力的计算工具，后续章节将对此加以讲解。

4.5.3　围绕中心位置的旋转操作

当对运动对象进行编程时，常需要围绕其中心位置执行旋转操作。这里，对象的中心位置常称作中心或质心，当工作于三维环境下时，这一概念十分重要。针对二维三角形，该点称作中心。如图 4.20 所示，中心位置位于 3 条直线的交点处，对应直线连接顶点和对边中点。

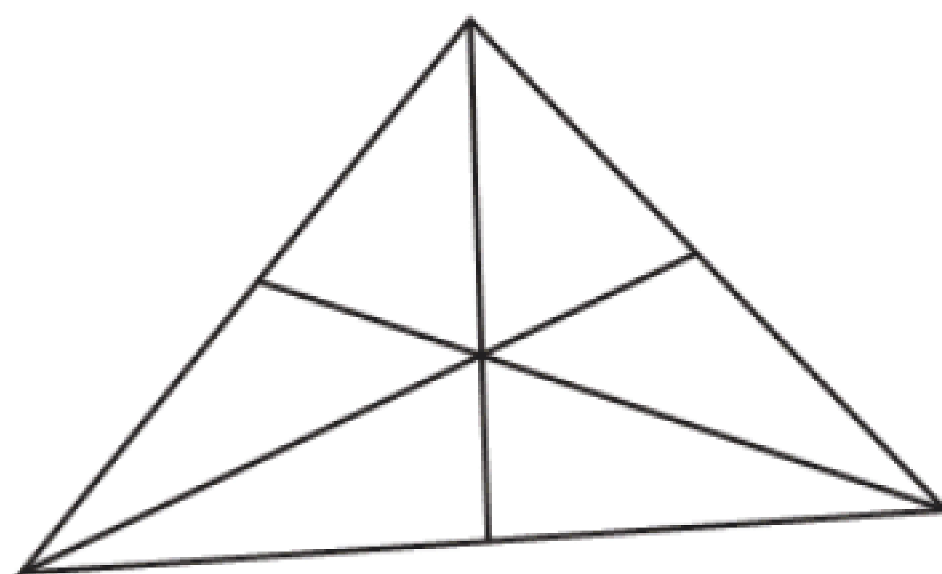


图 4.20　三角形中心

为了计算中心位置，须计算 3 个顶点的平均数。若 3 个顶点分别为(x_1, y_1)，(x_2, y_2)和(x_3, y_3)，则中心位置表示为$\frac{1}{3}(x_1+x_2+x_3, y_1+y_2+y_3)$。读者可参考练习 4.2，进而尝试计算三角形的中心位置。

数字 a_1，a_2，…，a_n 的算术平均数表示为数字之和除以数字数量，即$\frac{a_1+a_2+\ldots+a_n}{n}$，并可记为$\frac{1}{n}\sum_{i=1}^{n}a_i$ 符号项。其中，大写希腊字母“Σ”表示为基于索引 i 的数值之和。平均数也称作平均值，但数学家更偏爱使用平局值体现多个不同的功能，其中包括：算术平均数、几何平均数、调和平均数、中位数以及众数（mode），且应用于不同的计算场合。

4.5.4　基于特定角度值的快速旋转

尽管对象可围绕任意轴旋转，但读者应谨记某些结论性内容。基于特定角度的旋转通常较为简单，如下所示：

- 围绕原点旋转点(x, y)旋转 180°：仅需将两个坐标乘以–1 即可，即$(-x, -y)$。
- 围绕原点旋转点(x, y)旋转 90°：可将坐标切换为$(y, -x)$。
- 围绕原点旋转点(x, y)旋转–90°：可将坐标切换为$(-y, x)$。

上述结果源自第 3 章中的相关方程。

4.5.5　反射

如前所述，反射行为可根据平面上的一条直线确定，该直线称作反射轴。如图 4.21 所示，点 P 相对于轴 A 的反射结果表示为 P'，并满足 AP = AP'且 PP'垂直于 A 这两个条件。

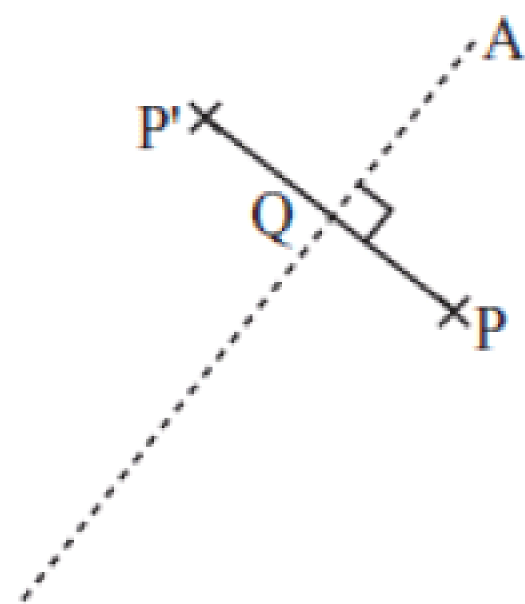


图 4.21　轴 A 上基于点 P 的反射操作

虽然可根据上述信息计算 P'的位置，但若缺乏向量的支持，则计算过程通常难以实现。然而，读者还可采用旋转和平移将其归结于一类简单问题。基于 X 或 Y 轴的反射操作较为简单，相应地，X 轴的反射点(x, y)将生成点$(x, -y)$，而 Y 轴的反射点则生成$(-x, y)$。因此，当采用方程 $y = mx + c$ 对某一轴的 $P = (x, y)$执行反射操作时，读者可尝试执行下列步骤（如图 4.22 所示）：

- 在 y 方向上平移 P 点$-c$ 个单位，则可得到 $P_1 = (x, y - c) = (x_1, y_1)$。
- 围绕原点旋转 P_1 点 atan(m)角度，则可得到：

$$P_2 = (l\cos(\text{atan}(m) - \text{atan}(y_1, x_1), l\sin(\text{atan}(m) - \text{atan}(y_1, x_1) = (y_2, x_2)$$

其中，$l=\sqrt{(x_1^2 + x_1^2)} = \sqrt{x^2 + (y+c)^2}$ 表示为直线 OP'的长度。

- 在 x 轴中反射 P_2 后可得到 $P_2 = (x_2, -y_2) = (x_3, y_3)$。
- 围绕原点 2 旋转 P_3 点$-$atan(m)角，则可得到如下算式：

$$P_4 = (l\cos(-\text{atan}(m) - \text{atan}(y_3, x_3)), l\sin(-\text{atan}(m) - \text{atan}(y_3, x_3))) = (x_4, y_4)$$

- 在 y 方向上平移 P_4 点 c 个单位，可得到 $P' = (x_4, y_4 + c) = (x', y')$。

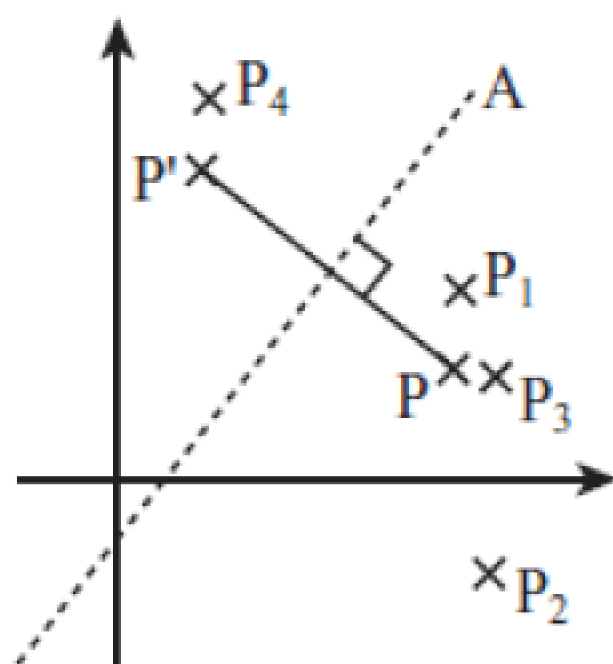


图 4.22　经一系列转换后得到 P'

当然，上述过程可通过三角恒等式实现进一步的简化，并合并为单一公式，此处并不打算对具体过程进行深入讨论。

除此之外，当使用向量和矩阵时，全部工作的操作难度将有所降低。

4.5.6　sin()、cos()和圆周运动

尽管引入三角形函数可降低计算难度，但这并非仅是一种数学抽象概念，它体现了某些自然现象。假设点 P 距圆心位置 1 个单位，如图 4.23 所示，根据 sin()和 cos()函数的定义可知，点 P 的坐标可表示为$(\cos(a),\sin(a))$。其中，a 为直线 OP 与 X 轴之间的夹角。若绘制全部点 P，则可

得到围绕原点的圆。

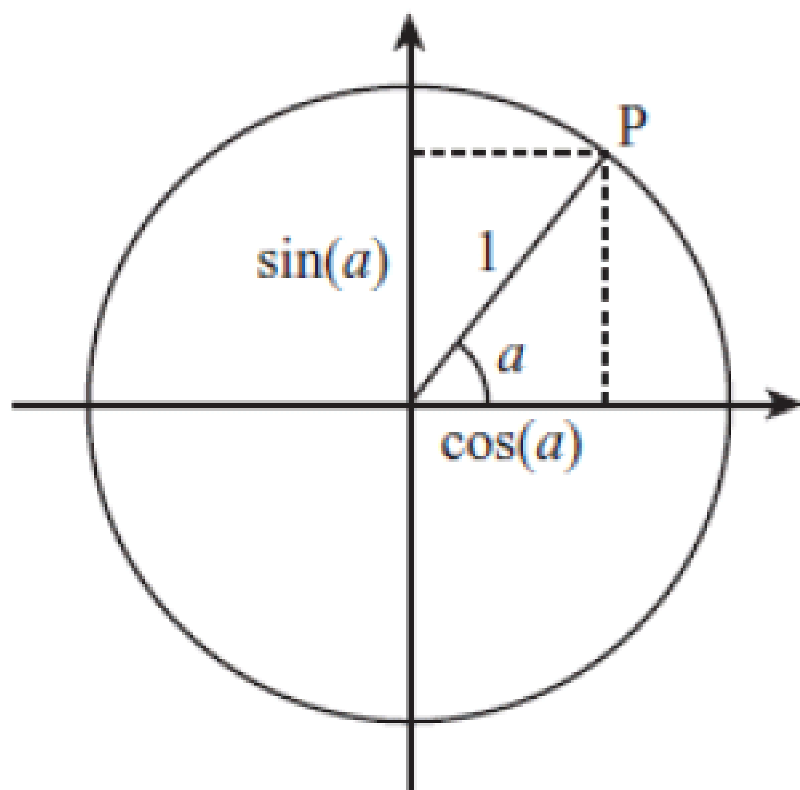


图 4.23　圆上的 sin()和 cos()函数

【提示】需要注意的是，直线 OP 的长度为 1，这也体现了前述恒等式 $\sin^2\theta + \cos^2\theta = 1$ 的正确性。

当使用 sin()和 cos()函数绘制圆时，点的位置将以恒定速度围绕圆周运动。若初始速度为水平方向，若向轮胎一侧钉入一枚图钉，则在轮胎转动过程中图钉垂直方向上的运动呈现为 sin()函数形状。在第 16 章讨论振荡现象时，将对此予以深入分析。当前，读者仅需了解位于半径为 r、圆心为(x, y)的圆上点，其左边表示为$(r\sin(\alpha), r\cos(\alpha))$。

4.6　本 章 练 习

【练习 4.1】试编写 solvetriangle(triangle)函数，该函数接收一个数组参数，包含了部分三角形信息，并返回一个包含详细信息的数组。

具体而言，solvetriangle(triangle)函数接收一个 6 元素数组，其中，前 3 个元素表示为三角形的边数据，而后 3 个元素表示为 3 个角度信息。取决于个人喜好，角度可采用角度或弧度制。另外，此类数据值可替换为“？”，并以此显示未知项。若三角形不存在，则函数返回 0；若三角形可求解，则函数返回 6 元素完整数组；若三角形不存在唯一解，则函数返回不完全数组。

【练习 4.2】试编写 rotatetofollow(triangle,point)函数，该函数围绕其中心位置旋转三角形，以使其对准某一既定点。

rotatetofollow(triangle,point)函数接收两个参数：数组表示三角形的 3 个顶点，另一个参数则表示对准的既定点。该时应可返回新顶点，若读者对此存在疑惑，则可在阅读完第 5 章后再次考察该练习。

4.7　本 章 小 结

本章讨论了与几何学相关的内容，以及大量的关键技术，并可用于动画和游戏中。第 5 章将对此予以深入讨论。

至此，读者应掌握如下内容：

- 角度的定义及其不同的计算方式。
- 面积的含义，以及如何计算矩形、圆形和三角形的面积。
- 三角形的类型及其属性，包括不等边三角形、等腰三角形、等边三角形、直角三角形。
- 如何使用三角函数和毕达哥拉斯定理求解直角三角形和其他三角形。
- 术语“相似”和“全等”的含义，以及各类形状的长度和角度。
- “旋转”、“反射”、“平移”、“缩放”和“剪切”的含义，以及针对既定点或平面形状旋转、反射和平移的计算方式。

第 5 章　向　　量

本章包含如下内容：

- 概述。
- 向量。
- 向量运动。
- 向量计算。
- 矩阵。

5.1　概　　述

本章将讨论向量的概念，及描述空间相对位置的数学对象。前述章节中曾使用了向量，本章将对此进行深入讨论。另外，本章最后还将讨论矩阵计算，并介绍如何通过矩阵实现空间的变化。

5.2　基 础 知 识

本章的第一个目标是讨论向量，并考察基于向量的基本计算。

5.2.1　“指令”向量

向量类似于指令，并告知对象的运动方向。例如，海盗探宝地图标明“向北行进 4 步，再向东行进 3 步，然后开始挖掘宝藏”，则可据此首先向北移动，并于随后向东行进。又如，饭店位置可描述为“穿过一楼走廊并在第一扇门处右转”，该指令描述了三维空间内的向量，即首先抵达一楼，穿越走廊并右转。

向量可用于描述移动方式，在海盗探宝地图示例中，若先向西一步，然后向北 4 步，最后再向东行进 3 步，只要起始点位于正确位置，则依然可抵达宝藏。需要注意的是，向量并不包含位置信息，且仅告知读者的起始点和结束点。

在本书中，向量采用黑体表示，例如 **u** 和 **v**。当工作于笛卡儿坐标系中时，可通过 x 和 y 方向上的移动距离标定某一向量，此类数据可借助形如 $\begin{pmatrix} x \\ y \end{pmatrix}$ 的阵列予以提供。例如，图 5.1 中的向量可采用 $\begin{pmatrix} -3 \\ 2 \end{pmatrix}$ 加以定义。由于对象沿负 x 方向运动，因而向量的 x 位置值为负值，这一类数据

值称为坐 x、y 方向上的向量分量。

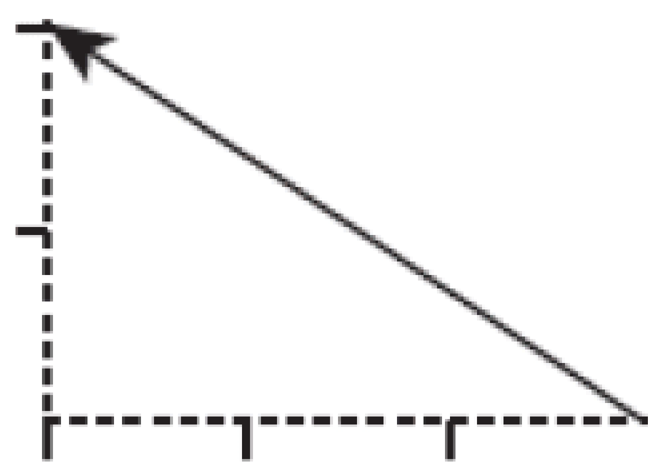


图 5.1　向量 $\begin{pmatrix} -3 \\ 2 \end{pmatrix}$

向量包含两个属性，即数值和方向。向量 **v** 的大小表示为空间长度，记为|**v**|，并可通过毕达哥拉斯定理计算。例如，向量 $\begin{pmatrix} x \\ y \end{pmatrix}$ 的大小为 $\sqrt{x^2+y^2}$。在二维空间中，可通过向量与轴向之间的夹角表示方向。若向量的大小为 1，则该向量称作单位向量，在二维空间中，针对某一角度 α，该向量等于 $\begin{pmatrix} \sin(\alpha) \\ \cos(\alpha) \end{pmatrix}$。

【提示】某些读者将向量的大小记为|v|，即不采用黑体书写向量。为了避免混淆，本书并未采用这一记法。

若针对某一向量确定了一个起始点，则该点称作位置向量。例如，在位置向量指令中，根据橡树位置向北行进 3 步，此处，橡树即为起始点。从数学角度来看，读者通常可选取标准的起始点。例如笛卡儿平面中的原点，并于随后根据该原点计算全部位置向量。如图 5.2 所示，若绘制源自原点的位置向量，则端点左边与坐标分量保持一致；如果端点标记为 O 和 P，则对应向量可记为 $\overline{\mathrm{OP}}$。

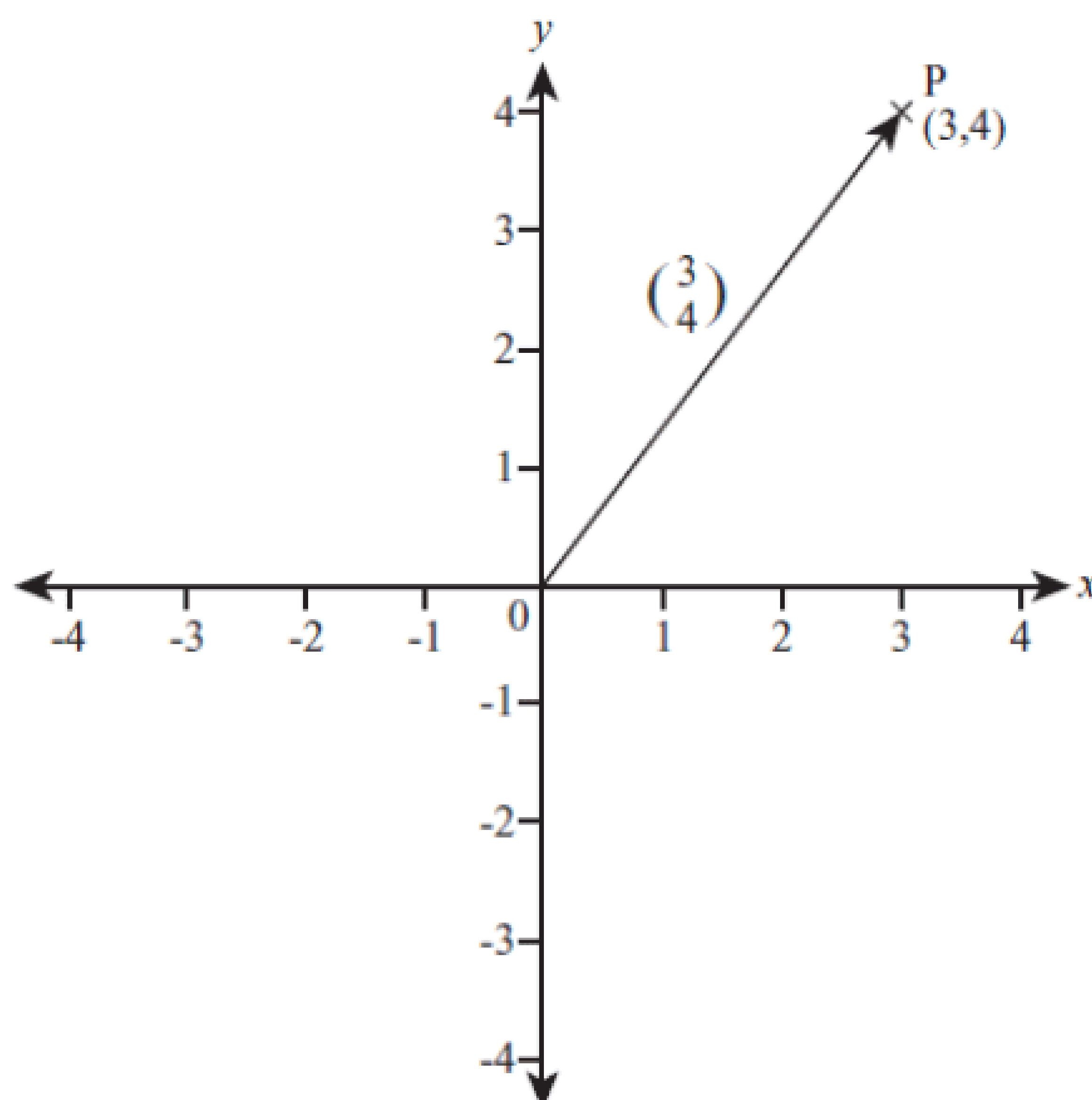


图 5.2　位置向量

脚标常用于表示向量的分量，例如，向量 **v** 包含分量(v_1, v_2)，这与程序设计中基于索引的数组分量十分类似。在程序设计语言中，数组 **v** 的分量可通过 v[0]，v[1]，v[2]等进行计算。需要注意的是，数组的第一项数据常采用索引 0 表示。在本书中，数组首项数据通过 v[1]表示，而非 v[0]。

5.2.2　向量算术

尽管存在两种操作可执行乘法运算，但两个向量之间无法通过简单方式相乘。本章和第 16 章介绍了两种计算方法，第 3 种方法常用于程序设计中，即对值乘法，即分量之间两两相乘，进而得到一个新向量。对值方法并非是标准的数学根据，该方法常用于类向量对象中，例如颜色数据，第 19 章将对此予以介绍。

1. 标量乘法和加法

与乘法相比，向量的加法则相对直观。类似地，读者还可方便地执行向量与标量之间的乘法运算。回忆一下，相对于数组和向量，标量值表示为单一值。

针对于向量加法，可在分量之间两两相加，如下所示：

$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}$$

在向量与标量的乘法运算中，各分量与标量逐一相乘，如下所示：

$$a\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax \\ by \end{pmatrix}$$

在图 5.3（a）中，向量与标量之间的乘法将改变向量的长度，且方向保持不变。其中，向量 **v** 乘以 2 后将得到 2**v**。若标量为负值，则结果向量将改为相反方向。需要注意的是，若向量 **v** 乘以−1，则结果向量变为−**v**，对应的箭头也应予以逆置。

除了逆置箭头之外，还可通过至少两种方式指定负变化。例如，若向量 $\overline{\text{AB}}$ 表示为 **v**，则负向量表示为 $\overline{\text{BA}}$ 。其中，箭头方向保持不变，而顶点顺序则发生了变化。另一种方法则是使用负号，即向量 **v** 的负向量表示为−**v**。

针对于非 0 向量，若该向量除以|**v**|或乘以其大小的倒数，对应结果将得到单位向量。单位向量有时采用 $\overline{\mathbf{v}}$ 或 $\hat{\mathbf{v}}$ 表示，常称作 **v** 的标准化向量或范数。

图 5.3（b）显示了两个向量之和（回忆一下，向量并不关心相对于端点的路线，仅关注最终位置），其中，**u** 和 **v** 分别表示不同的方向，二者之和表示两个向量的到达点。

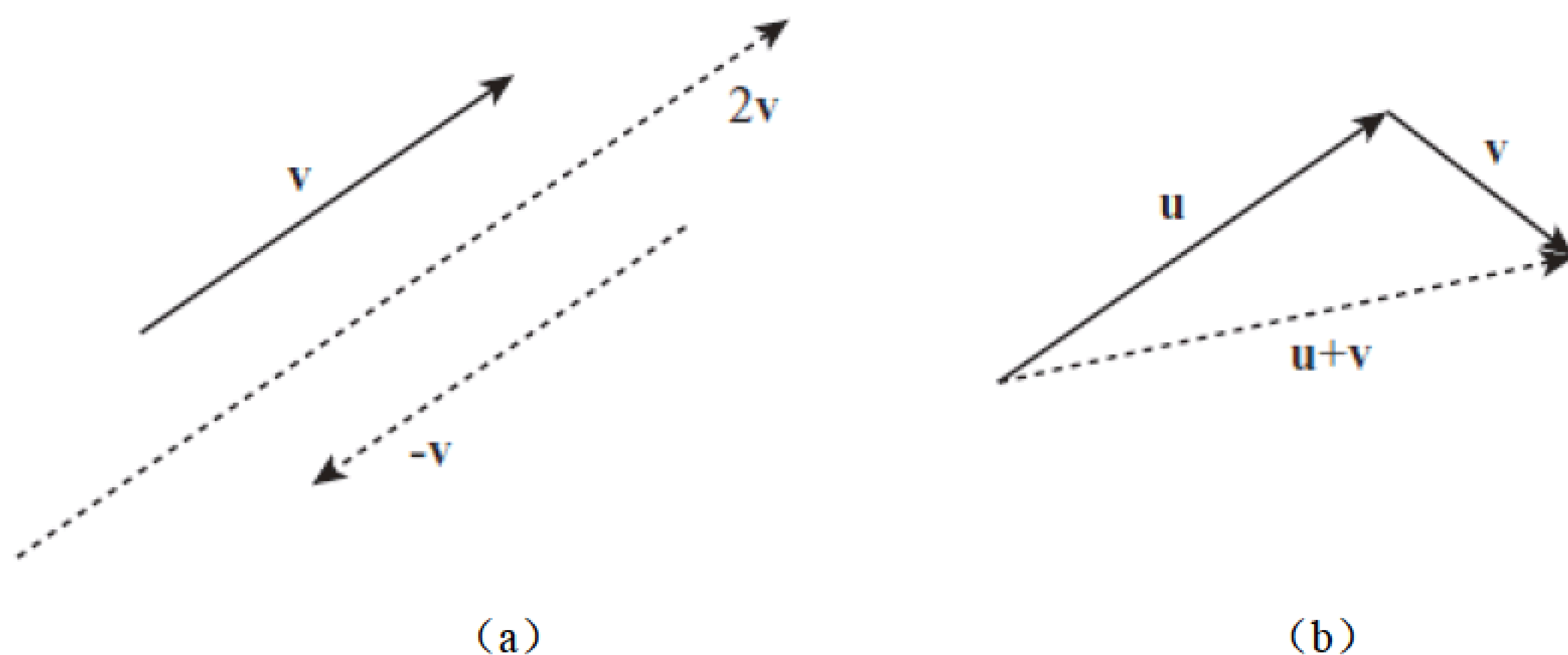


图 5.3　向量的加法以及向量与标量之间的乘法运算

2. 向量的减法

与向量的加法以及向量与标量之间的乘法运算相比，向量的减法则稍显复杂，如图 5.4 所示。其中，若向量 **u** 和 **v** 分别表示位置向量，即 P 和 Q，且有 **u**=$\overrightarrow{OP}$，**v**=$\overrightarrow{OQ}$，二者之间的差 **v**−**u** 等于 OP。

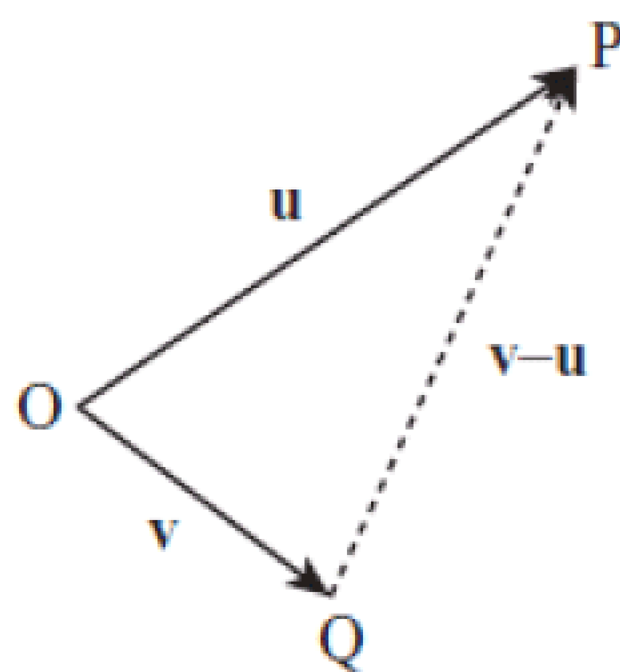


图 5.4　两个向量之差

若初始点源于点 P，并分别沿 −**u** 和 **v** 行进，则最终可抵达 Q，因而有 **v**−**u**=−**u**+**v**=$-\overrightarrow{OP}+\overrightarrow{OQ}=\overrightarrow{PO}+\overrightarrow{OQ}=\overrightarrow{PQ}$。

向量具有强大的计算功能，下面考察图 5.4 所示的直线中点的位置向量。若移动行始于点 O 并移至点 P，则沿 PQ 行进至半途时即可到达中点 M。向量 $\overrightarrow{OM}$ 等于 $\overrightarrow{OP}+\frac{1}{2}\overrightarrow{PQ}=\mathbf{u}+\frac{1}{2}(\mathbf{v}-\mathbf{u})=\frac{\mathbf{u}+\mathbf{v}}{2}$，即 **u** 和 **v** 的平均数。类似地，三角形中心的位置向量可表示为其顶点位置向量的平均数。

5.2.3　向量编程

某些程序设计语言支持向量计算，可添加数组并使其与标量进行乘法运算。3D 引擎通常可计算向量的大小和范数，下列函数集显示了此类函数的实现方式。

addVectors()函数执行向量 **v1** 和 **v2** 之间的加法运算，如下所示：

```
function addVectors(v1, v2)
  //assume v1 and v2 are arrays of the same length
  set newVector to an empty array
  repeat for i=1 to the length of v1
  append v1[i]+v2[i] to newVector
  end repeat
  return newVector
end function
```

scaleVector()函数执行向量 **v** 和因子 s 之间的乘法运算，如下所示：

```
function scaleVector(v, s)
  repeat for i=1 to the length of v
    multiply v[i] by s
```

```
    end repeat
    return v
end function
```

magnitude()函数计算向量 **v** 的大小，如下所示：

```
function magnitude(v)
  set s to 0
  repeat with i=1 to the length of v
    add v[i]*v[i] to s
  end repeat
  return sqrt(s)
end function
```

norm()函数执行向量的标准化计算，如下所示：

```
function norm(v)
  set m to magnitude(v)
  //you can't normalize a zero vector
  if m=0 then return "error"
  return scaleVector(v,1/m)
end function
```

下面考察一类较为重要的计算，即两个向量之间的夹角，本章后续内容还将介绍一种较为简单的方法。在图 5.4 中，向量 **u**，**v** 和 **u** – **v** 构成了三角形 OPQ。对此，读者可采用余弦定理并根据向量的大小计算任意角度值。特别地，可通过下列方式计算 **u** 和 **v** 之间的夹角 θ：

$$\cos\theta = \frac{|\mathbf{u}|^2 + |\mathbf{v}|^2 - |\mathbf{u}-\mathbf{v}|^2}{2|\mathbf{u}||\mathbf{v}|}$$

对应函数如下所示：

```
function angleBetween(vector1, vector2)
  set vector3 to vector2-vector1
  set m1 to magnitude(vector1)
  set m2 to magnitude(vector2)
  set m3 to magnitude(vector3)
  //it makes no sense to find an angle with a zero vector
  if m1=0 or m2=0 then return "error"
  if m3=0 then return 0 //the vectors are equal
  return acos((m2*m2+m1*m1-m3*m3)/(2*m1*m2))
end function
```

5.2.4　法向量

若两个向量彼此垂直，则称其处于法向关系。在二维环境中，可方便地计算既定向量 $\begin{pmatrix} a \\ b \end{pmatrix}$ 的垂直向量。对此，可反转当前向量并获取某一分量的负值，即 $\begin{pmatrix} -b \\ a \end{pmatrix}$。这里，可将向量视为某一

位置向量，并采用 90° 的快速旋转操作。由于向量的标量倍数仍然垂直于原向量，因而可指定任意负分量。相应地，乘以–1 后将得到另一方向上的同一向量。对应函数的实现过程如下所示：

```
function normalVector(vector)
   return vector(-vector[2],vector)
end function
```

当且仅当分量乘积之和为 0，则两个向量处于垂直状态。对于 normalVector()函数，各向量 x 分量的乘积为$-ab$，y 分量的乘积为 ab，因而其和为 0，稍后将对此进行深入讨论。

5.2.5　真实世界中的向量和标量

在日常生活中，许多量值均可通过向量进行测算，本小节将对此加以介绍，进而了解向量的含义及其重要性。当使用向量时，应用于通用环境下的多个术语常呈现出特定的含义，如下所示：

- 距离。距离定义为标量值，用于计算两点间最短直线的长度。
- 位移。若定义了由向量连接的两点，则该向量体现了两点间的位移。
- 速率。作为标量值，速率用于测算既定时间内对象行进的距离。
- 速度。向量用于确定速度，速度表示既定时间内的位移。
- 质量。作为一个标量值，质量体现了移动对象时的作用力程度，且与运动方向无关。
- 重量。重量定义为一个向量，即特定方向上的作用力，进而抵消重力以使对象位于同一位置。

在某些非正式场合，距离和位移、速度和速率、质量和重量往往交互使用。例如，当描述对象的质量时，常会谈到该对象具有一定的重量。又如，火车以 100km/h 的速度行驶，实际上，该值表示为火车的速率。对于大多数人来讲，向量并非是一种思维定式。

需要注意的是，在当前上下文环境中，读者应区分技术用途和非技术用途之间的差别。稍后将会看到，当使用牛顿定律时，将会面临改变对象速度的作用力。对于某一对象，其速度发生变化，而速率保持不变。例如，考察绑定于绳索一端的球体对象，若球体做圆周运动，其运动方向不断改变，但其速率却保持恒定。当采用向量描述这一现象时，其行为将变得较为简单，其中，对象的速度定义为向量，其速率则表示为速度的大小。因此，球体对象的速度处于变化状态，而速率保持不变。

5.3　基于向量的运动

本小节介绍如何通过向量进行相关计算，后续章节将对此进行适当的扩展，尤其是处理碰撞检测及其解决方案时。

5.3.1　通过向量描述形状

向量可方便地描述平面点之间的关系，例如平行线。若两条直线间不存在交点，或二者拥有

无数个交点（此时，两直线为同一条直线），则称两条直线处于平行状态。然而，对于包含端点的直线（即线段），具体处理过程则稍显复杂。

1. 平行线

当使用向量时，对应处理可得到一定程度的简化。对于一条直线上的两点 P 和 Q，以及另一直线上的两点 P′ 和 Q′，存在某一标量值可满足 $\overrightarrow{PQ} = a\overrightarrow{P'Q'}$。

2. 正方形

向量还提供了与平面形状生成相关的处理方案。例如，在图 5.5 中，若给定包含位置向量 **a** 和 **b** 的两点 A 和 B，则可绘制一个正方形。首先需要计算 $\overrightarrow{AB}$（即 **b–a**）的法线 **n**，且与 **b–a** 具有相同的大小，否则，则需要执行相应的缩放操作。随后，可采用 **c=a+n** 和 **d=b+n** 构造点 C 和 D。根据上述预计算，点 A，B，C，D 构成了一个正方形。

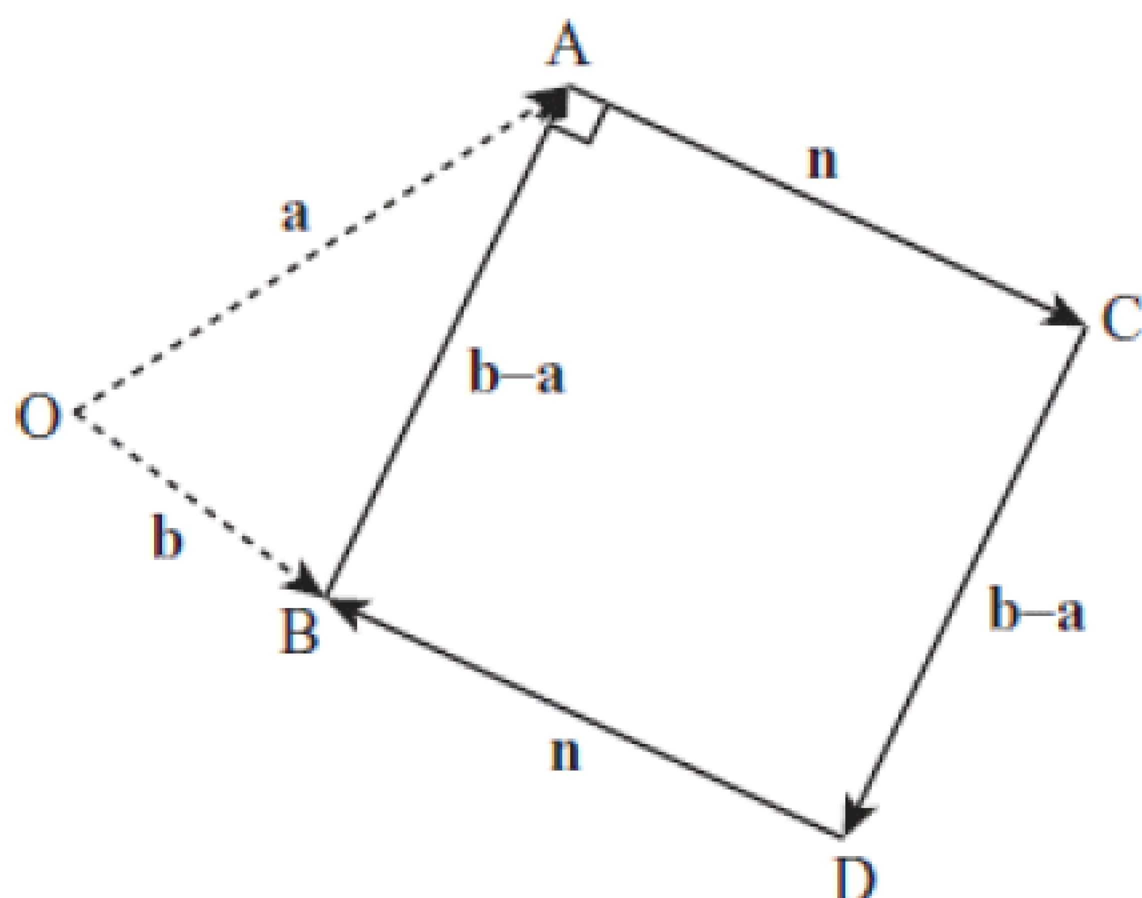


图 5.5　利用向量构造正方形

【提示】需要注意的是，取决于法线向量 **n** 的选取方向，图 5.5 可能存在两个正方形。

3. 等边三角形

类似于正方形，等边三角形的构造过程同样简单。该构造过程涉及少量的三角形内容，其中，等边三角形顶点至对边中点的线段长度为 $\frac{\sqrt{3}}{2}$ 倍的边长（该结果可通过毕达哥拉斯定理予以证明）。若包含两个顶点 A 和 B，则可通过 $\frac{\mathbf{a}+\mathbf{b}}{2}+\frac{\sqrt{3}\mathbf{n}}{2}$ 构造第 3 个顶点 C。这里，**n** 表示为 **b–a** 的法向量。回忆一下，$\frac{\mathbf{a}+\mathbf{b}}{2}$ 表示为 A 和 B 中点的位置向量。

4. 其他形状和函数

相应地，读者还能通过类似方法构造其他复杂形状。在练习 5.1 中，读者可尝试编写一组函数，并构造相应的形状，例如箭头或风筝形。此类函数的优势在于，可对其实现参数化垂直，进而生成同一形状的多个变体。下列函数可生成字母 A 的不同字体：

```
function createA(legLength, angleAtTop, serifProp, crossbarProp,crossbarHeight,
                 serifAlign, crossbarAlign)
```

```
  //serifProp, crossbarHeight, crossbarProp,
  //serifAlign and crossbarAlign should be values from 0 to 1
  //angleAtTop should be in radians
  set halfAngle to angleAtTop/2
  set leftLeg to legLength*array(-sin(halfAngle), cos(halfAngle))
  set rightLeg to array(-leftLeg, leftLeg[2])
  set crossbarStart to leftLeg*crossbarHeight
  set crossbarEnd to rightLeg*crossbarHeight
  set crossbar to crossbarProp*(crossbarEnd-crossbarStart)
  add crossbarAlign*(1-crossbarProp)*(crossbarEnd-crossbarStart) to crossbarStart
  set serif to serifProp*(rightLeg-leftLeg)
  set serifOffset to serifAlign*serif
  set start to array(0,0)
  drawLine(start, leftLeg)
  drawLine(start, rightLeg)
  drawLine(crossbarStart, crossbarStart+crossbar)
  drawLine(leftLeg-serifOffset, leftLeg-serifOffset+serif)
  drawLine(rightLeg-serifOffset, rightLeg-serifOffset+serif)
end function
```

图 5.6 显示了基于 createA()函数的字母绘制过程，并可通过对应参数生成不同风格的字体。

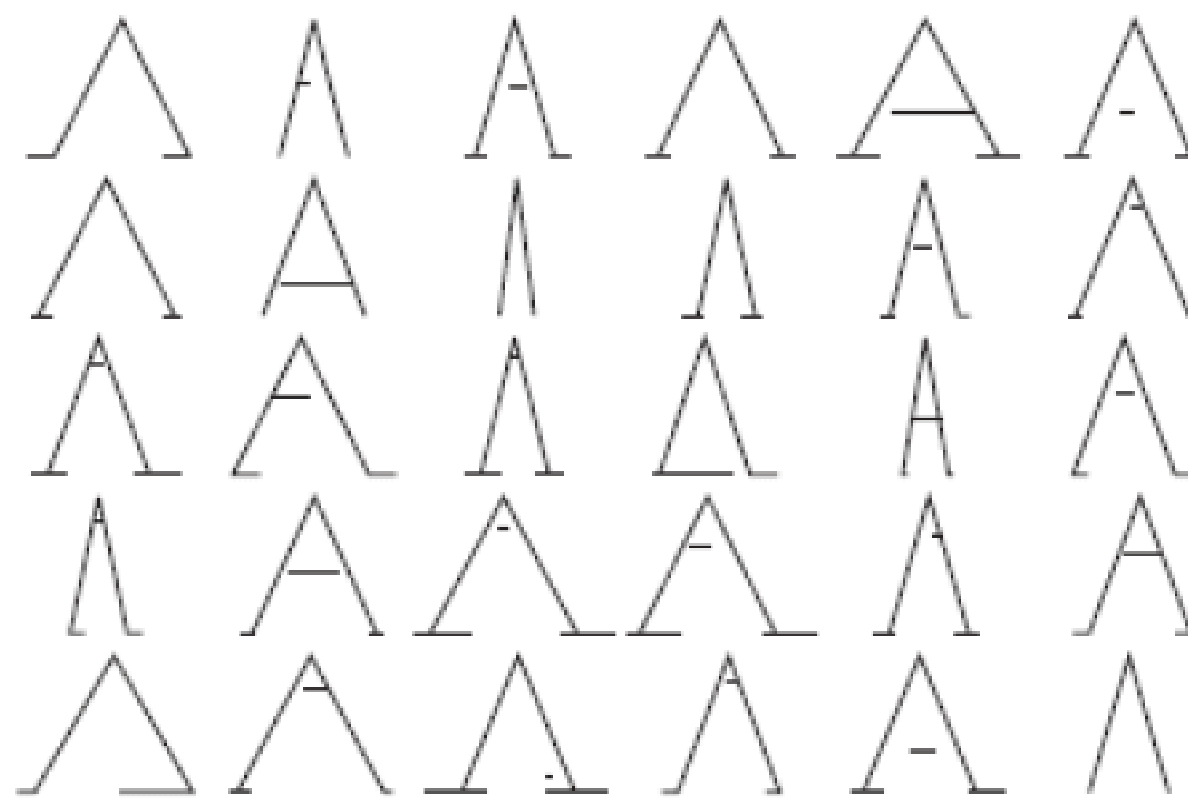


图 5.6　利用 createA()函数绘制字母 A

5.3.2　P 和 Q 之间的运动

截止到目前为止，本章所讨论的向量内容构成了程序设计中的一类基础问题，这对于游戏开发而言十分重要。在图 5.7 中，人物角色从点 P 移至点 Q，假设该游戏角色为男性且名为 Jim，若 Jim 位于(a, b)并移至(c, d)，则如何表示对应的运动路径？

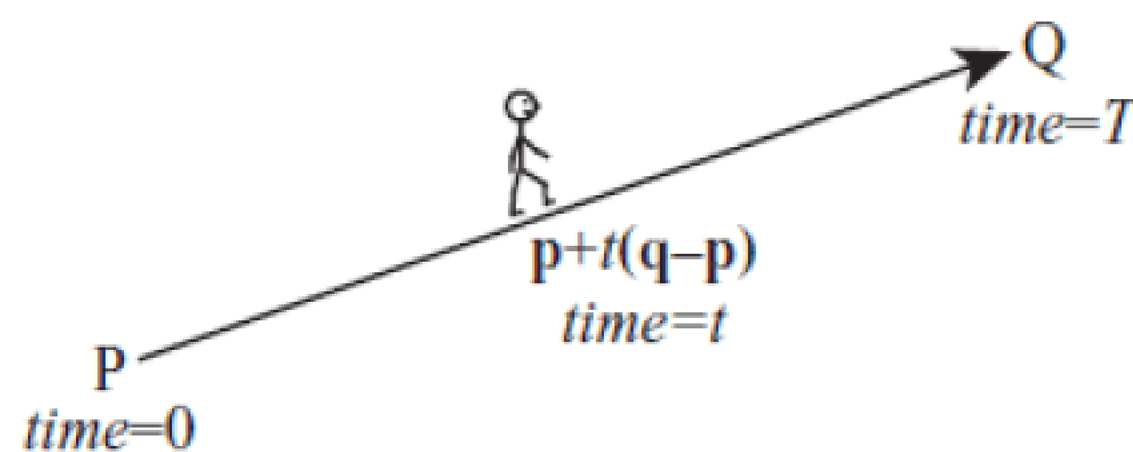


图 5.7　Jim 的行进路径

为了求解该问题，可将其分解为多个步骤，如下所示：

- 时刻 0 处，Jim 位于 $P=(a,b)$。
- 时刻 T 处，Jim 位于 $Q=(c,d)$。
- 此处需要求解时刻 t 处的左边，且有 $0\leqslant t\leqslant T$。

对此，有必要引入速率和速度这两个概念，相应地，可通过向量方式求解该问题。在时间 T 内，Jim 沿向量 $\overrightarrow{PO}$（即 $\begin{pmatrix} c \\ d \end{pmatrix}-\begin{pmatrix} a \\ b \end{pmatrix}=\begin{pmatrix} c-a \\ d-b \end{pmatrix}$）直线行进，即位移 z，而向量的长度 $\sqrt{(c-a)^2+(d-b)^2}$ 则表示为 Jim 行进的距离。

距离除以时间即可得到速率，相应地，可采用距离单位除以时间单位加以表示，例如，米/秒或 ms^{-1}。这里，速率表示为各时间单位内的行进距离，速度则表示为位移向量除以时间值后的结果，即各单位时间内的行进向量。据此可知，Jim 的速度可表示为 $\frac{1}{T}\begin{pmatrix} c-a \\ d-b \end{pmatrix}$。

下面考察 Jim 于时刻 t 时的位置。对此，可查看沿向量 $\overrightarrow{PQ}$ 的比例值 t/T。由于 $t=mT$，因而可将该比例值表示为 m，因而 Jim 的位置向量如下所示：

$$\overrightarrow{OP}+m\overrightarrow{PQ}=\begin{pmatrix} a \\ b \end{pmatrix}+m\begin{pmatrix} c-a \\ d-b \end{pmatrix}=\begin{pmatrix} mc+(1-m)a \\ md+(1-m)b \end{pmatrix}$$

【提示】 此处，速率显得较为微妙。若某一角色沿圆周运动并最终停止于初始点，则该过程中的位移为 0，因而速度以及平均速度皆为 0。然而，在行进过程中，平均速率则不为 0，该值等于圆周长（行进距离）除以时间值。也就是说，速率通过行进路径的长度需计算，而非结果向量的长度。当沿直线行进时，二者间的区别则并不明显。

当对运动行为编程时，通常需要预计算某些数据值。在面向对象程序设计中，往往会采用特定对象表示屏幕上的精灵对象，该对象接收位置和时间参数，并在两个位置之间移动。calculateTrajectory()函数即实现了此项任务，如下所示：

```
function calculateTrajectory(oldLocation, newLocation, travelTime)
  if time=0 then
    justGoThere(newLocation)
  otherwise
    set displacement to newLocation-oldLocation
    set velocity to displacement/travelTime
    set startTime to the current time
    set stopPosition to newLocation
    set startPosition to oldLocation
  end if
end function
```

待运动轨迹计算完毕后，若意欲更新精灵对象的位置，则可直接计算对应的新位置。currentPosition()函数用于实现这一任务，如下所示：

```
function currentPosition()
  set time to the current time-startTime
  if time>travelTime then
```

```
    set current position to stopPosition
  otherwise
    set current position to startPosition+velocity*time
  end if
end function
```

总体而言，当确定某一对象的当前位置时，一类较好的方法则是获取该对象的标准速度，并计算相对于此刻的行进时间，练习 5.2 即使用了这一处理方案。

5.3.3 复杂的向量路径

向量运动并非仅用于直线，前述内容曾探讨了基于一系列向量的简单形状的构造方式。本小节将对此予以适当扩展，并讨论粒子速度变化时的曲线包围路径。

【提示】针对具有不确定运动方式的对象，粒子可视为其数学简称。尽管粒子具有电荷以及质量等属性，但通常假设为无穷小。在程序设计中，粒子往往用于描述屏幕上的运动元素（精灵对象）。

下面考察如图 5.8 所示的运动行为。在左图中，Jim 围绕点 Q 运动，并向其移动轨迹加入多个法向量。相应地，Jim 并未沿 PQ 直接移动，相反，该角色分别沿 PQ 及其垂直方向运动至 P'。Jim 按照相同方式移至 P''。

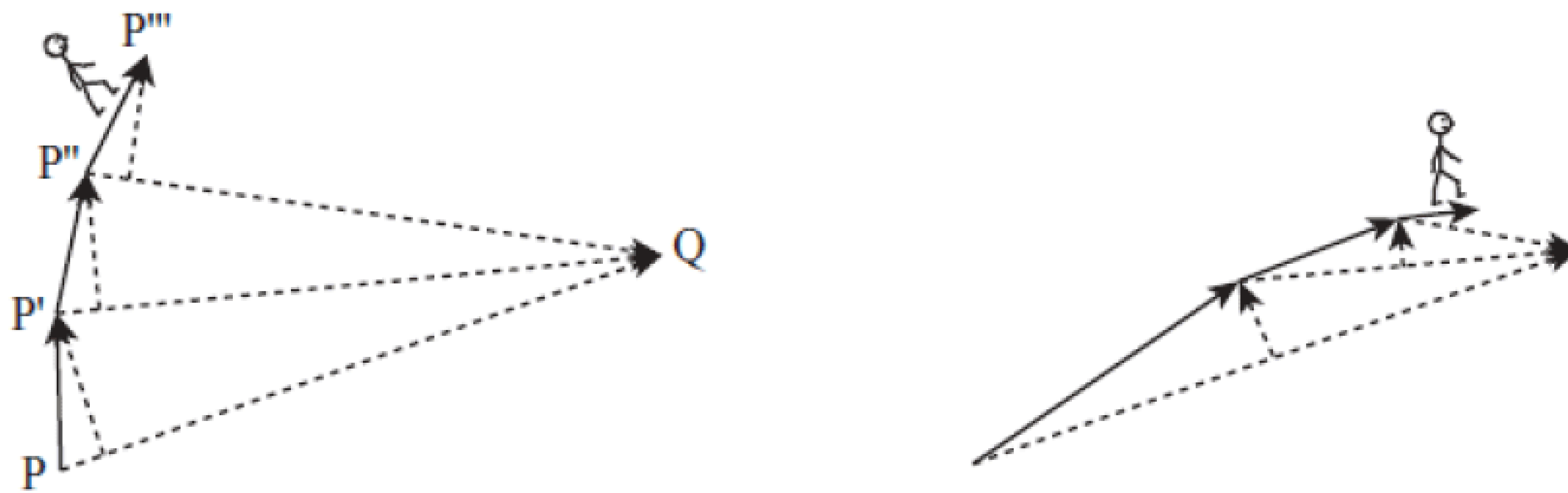


图 5.8　Jim 避开点 Q

对上述运动行为稍作扩展即可发现，Jim 的运动路径呈螺旋线形状，如图 5.8 中的右图所示。其中，螺旋线的紧密程度取决于法向量的大小（相对于内向速度）。若切向分量为 0，则 Jim 沿直线行进；若切向分量较小且大于 0，则角色沿弯曲路径运动；若切向分量较大，则当前角色以渐进式螺旋线方式移动；若切向分量无穷大，则 Jim 围绕点 Q 做圆周运动。

总体而言，由于运动粒子在屏幕上的时间步并非无穷小，因而从数学角度上讲，粒子的运动路径并不精确。尽管如此，当前方案已然可模拟 Jim 的运动行为，curvedPath()函数显示了基于曲线路径的粒子运动，如下所示：

```
function curvedPath(endPoint, currentPoint,speed, normalProportion, timeStep)
  set radius to endPoint-currentPoint
  if magnitude(radius)<speed*timeStep then
    set current position to endPoint
```

```
    otherwise
      set radialComponent to norm(radius)
      set tangentialComponent to
          normalVector(radialComponent)*normalProportion
      set velocity to speed*norm(radialComponent+tangentialComponent)
      set current position to currentPoint+velocity
    end if
  end function
```

5.3.4　奇异路径

当使用 curvedPath()函数时，normalProportion 值等于单位长度向量$\begin{pmatrix}\sin(\alpha)\\\cos(\alpha)\end{pmatrix}$，其中，$\alpha$ 等于 atan(normalProportion)。若尝试改变该向量，情况又当如何？例如，若围绕圆以恒定速率调整 α 值，以使该向量具有自身的“速度”。对应结果如图 5.9 所示。



图 5.9　变化速度生成的路径

针对上述路径，对应函数须调整各自的处理方案，madPath()函数即显示了一类解决方案，如下所示：

```
  function madPath(endPoint, currentPoint,currentAlpha, speed, alphaSpeed, timeStep)
    set radius to endPoint-currentPoint
    if magnitude(radius)<speed*timeStep then
      set current position to endPoint
    otherwise
      set radialComponent to norm(radius)
      set newAlpha to currentAlpha+alphaSpeed*timeStep
      set tangentialComponent to
           normalVector(radialComponent)*tan(newAlpha)
      set velocity to speed*norm(radialComponent+tangentialComponent)
      set current position to currentPoint+velocity
    end if
  end function
```

基于向量的抽象数据可视为一类强大的计算根据，但通常难以获得期望的结果。当前，读者仅需了解处于变化状态的速度向量，同时，这将导致加速度的出现。

5.4 向量计算

本小节将讨论向量方程的生成和求解方案。

5.4.1 向量及其分量

具有相同原点的非平行向量可用于描述平面内的任意一点。若 **u** 和 **v** 表示为非平行向量，则平面内任意向量均可采用 $a\mathbf{u}+b\mathbf{v}$ 这一形式加以描述。其中，a 和 b 为标量。当采用此方案时，**u** 和 **v** 用作坐标系，且常会使用到$\begin{pmatrix}1\\0\end{pmatrix}$和$\begin{pmatrix}0\\1\end{pmatrix}$，并采用 **i** 和 **j** 加以标识。由于向量$\begin{pmatrix}a\\b\end{pmatrix}$等于 $a\mathbf{i}+b\mathbf{j}$，因而向量分量直接转换为坐标系描述方式。鉴于此类坐标系向量彼此正交（即相互垂直）且为标准化向量，因而对应关系可描述为正交坐标系。

某些时候，其他类型的坐标系也十分有用。如图 5.10 所示，在实际操作过程中，还可考察径向和切向运动分量，这与直接指向目标的、基于正交系统坐标系的速度向量保持一致。

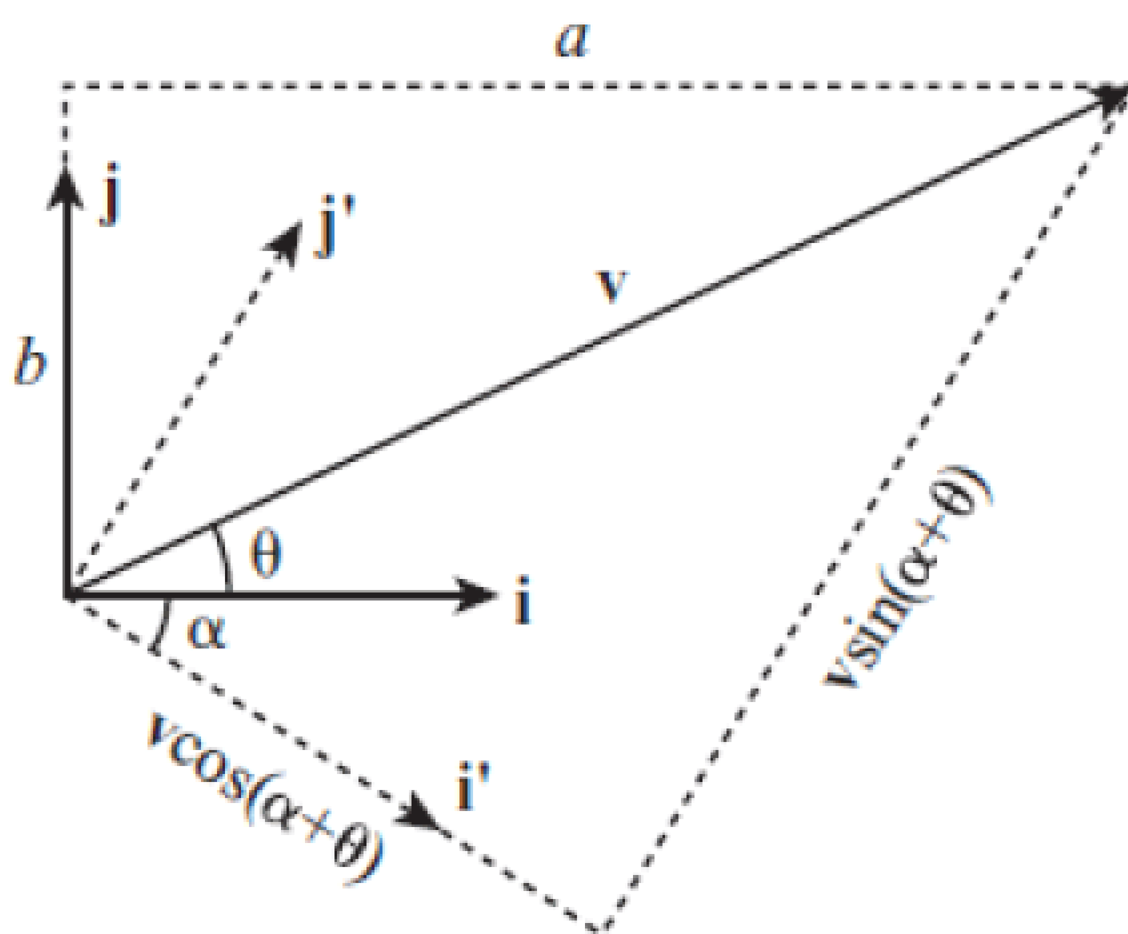


图 5.10　将向量转换为新坐标系

该方案的优点在于，可独立考察两个方向上的分量。如前所述，当作用力指向某一向量时，与该向量垂直的速度保持不变。

【提示】分量包含双重含义，针对正交坐标系，若有 $\mathbf{v}=p\mathbf{a}+q\mathbf{b}$，则可使用 a 方向上的 **v** 分量指定 $p\mathbf{a}$ 或数值 p——这通常可通过上下文加以判断。针对程序设计，读者可定义两个函数 component(vector1, vector2)和 componentVector(vector1, vector2)，进而对二者加以区分。

在图 5.10 中，向量 **v** 根据 **i**，**j** 转换为 **i'**和 **j**。**i** 和 **i'**之间的夹角表示为 α，**v** 和 **i** 之间的夹角表示为 θ。若围绕平行于新轴的向量绘制一个矩形，将会发现 **i'**方向上的分量值等于 $|\mathbf{v}|=\cos(\theta-\alpha)\begin{pmatrix}a\\b\end{pmatrix}=a\mathbf{i}+b\mathbf{j}$，**j'**方向上的分量等于$|\mathbf{v}|=\cos(\theta-\alpha)$。进一步讲，可直接根据 **i** 和 **j** 方向上

的 **v** 和 **i'**分量计算 θ 和 α 角。例如，若 **v** 表示为 $\begin{pmatrix} a \\ b \end{pmatrix} = a\mathbf{i} + b\mathbf{j}$，则 $\theta = \text{atan}(b, a)$。

针对实现函数，switchBasis()函数接收两个参数：向量 **v** 和 **k**，并返回 4 个值，即正交向量 **i'**和 **j'**（其中，**i'**为 **k** 的标准化版本）以及 **i'**和 **j'**方向上的 **v** 分量 a 和 b。最终，则可得到 $\mathbf{v} = a\mathbf{i}' + b\mathbf{j}'$。对应函数如下所示：

```
function switchBasis(vector, directionVector)
  set basis1 to norm(directionVector)
  set basis2 to normal(basis1)
  set alpha to atan(basis1[2],basis1[1])
  set theta to atan(vector[2],vector[1])
  set mag to magnitude(vector)
  set a to mag*cos(theta-alpha)
  set b to mag*sin(theta-alpha)
  return new array(basis1, basis2, a, b)
end
```

【提示】需要注意的是，此处使用了双参数的 atan()版本，该函数于第 4 章被引入。

当然，读者也可尝试编写两个相对简单的函数，以计算新坐标系中的独立分量。其中，首个函数 component()如下所示：

```
function component(vector, directionVector)
  set alpha to atan(directionVector [2], directionVector [1])
  set theta to atan(vector[2],vector[1])
  set mag to magnitude(vector)
  set a to mag*cos(theta-alpha)
  return a
end function
```

第二个函数 componentVector()如下所示：

```
function componentVector(vector, directionVector)
  set v to norm(directionVector)
  return component(vector, directionVector)*v
end function
```

根据前述讨论内容可知，与 switchBasis()函数相比，componentVector()和 component()函数可能更为常用。

5.4.2　标量积（点积）

截止到目前为止，尽管角度和向量分量计算较为简单，但依然存在另一种通用处理方案。如前所述，两个向量之间的乘法运算并不存在一类相对自然的计算方案。一类较为常见的操作将会涉及标量积运算。顾名思义，标量积整合两个向量并得到一个标量结果。

向量 **u** 和 **v** 的标量积记为 $\mathbf{u} \cdot \mathbf{v}$，其中，“·”使得标量积也称作点积。当计算标量积时，须计

算对应变量分量乘积之和，如下所示：

$$\begin{pmatrix} a \\ b \end{pmatrix} \cdot \begin{pmatrix} c \\ d \end{pmatrix} = ac + bd$$

此处，读者可查看既定向量与坐标系向量 **i** 之间的标量积结果，进而对其用途加以考察。其中，坐标系向量表示为$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$，并定义了该向量的 x 分量。坐标系向量与既定向量 **j** 之间的点积将生成 y 分量。类似地，向量 **v** 和任意单位向量之间的点积将得到 **u** 方向上的 **v** 分量。

两个向量 **v** 和 **w** 的点积等于$|\mathbf{v}|\times|\mathbf{w}|\times\cos\alpha$。其中 α 表示两个向量之间的夹角，这也意味着，可通过点积实现某些有意义的计算。例如，向量与其自身的点积表示为其值的平方（源自毕达哥拉斯定理）。同时，点积还可实现两个向量之间的角度计算，如下所示：

$$\alpha = \cos^{-1}\left(\frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|}\right)$$

点积运算包含如下特征：

- 点积符合交换律，即 $\mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{v}$。
- 点积符合加法的分配律，即 $\mathbf{v} \cdot (\mathbf{u} + \mathbf{w}) = \mathbf{v} \cdot \mathbf{u} + \mathbf{v} \cdot \mathbf{w}$。
- 针对标量乘法，则有 $\mathbf{v} \cdot (a\mathbf{u}) = a(\mathbf{v} \cdot \mathbf{u})$。
- 若两个变量彼此垂直，则二者的点积为 0，反之亦然。

5.4.3 向量方程

类似于常规数值，向量之间也可实现代数运算。实际上，当与第 3 章中的联立方程协同工作时，已然使用了向量计算。相应地，向量方程具有如下形式：

$$a\mathbf{u} + b\mathbf{v} = \mathbf{w}$$

方程中的任意标量均可为未知项，例如标量 a 和 b，或者向量 **u**，**v** 和 **w**。针对下列方程，较为常见的情况是：向量数据已知，而标量值未知。

图 5.11 为计算两条直线 AB 和 CD 之间的交点，其中，4 个顶点 A，B，C，D 的位置向量 **a**，**b**，**c**，**d** 为已知内容，并计算点 P 的位置向量，即两条直线的交点。

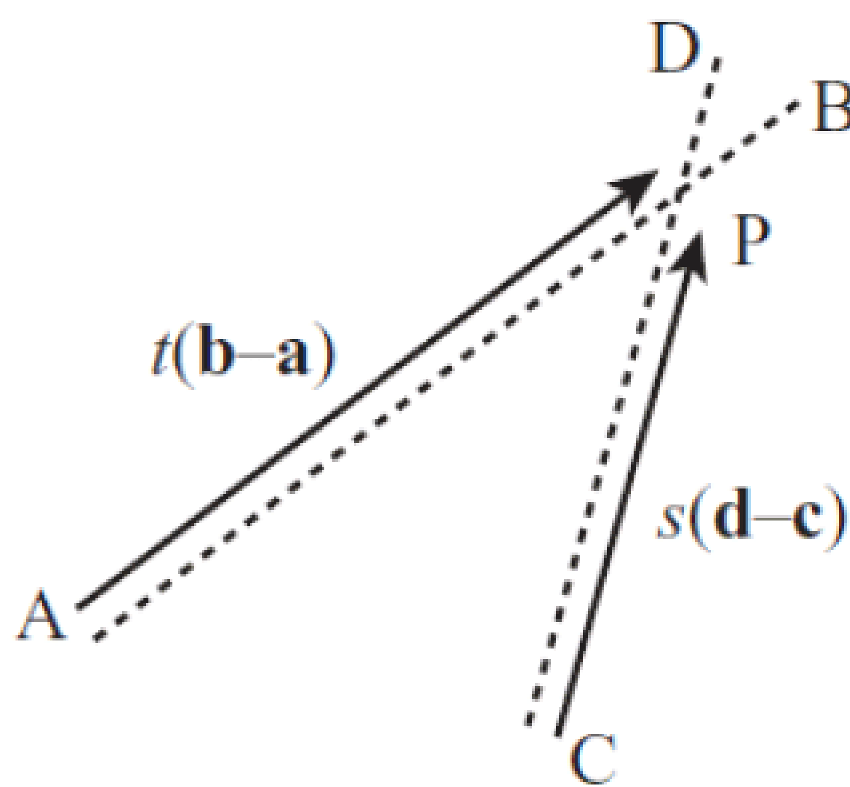


图 5.11　计算两条直线的交点

此处的关键技巧是对 P 执行参数化操作。换而言之，需要根据点 A，B，C，D 求出点 P。

当前，与 P 相关的全部已知内容是：该点位于直线 AB 和 CD 上。针对直线 AB 上的一点，可从原点出发并途经 A，进而沿向量 AB 行进一段距离。由于行进距离未知，因而可将其定义为 t，则当前问题如下所示：

$$\overrightarrow{OP}=\overrightarrow{OA}+t\,\overrightarrow{AB}=\mathbf{a}+t(\mathbf{b}-\mathbf{a})$$

由于 P 同时位于直线 CD 上，因而针对某一标量值 s，有如下算式：

$$\overrightarrow{OP}=\mathbf{c}+s(\mathbf{d}-\mathbf{c})$$

这将生成与 s 和 t 相关的等式，如下所示：

$$\mathbf{a}+t(\mathbf{b}-\mathbf{a})=\mathbf{c}+s(\mathbf{d}-\mathbf{c})$$

$$t(\mathbf{b}-\mathbf{a})+s(\mathbf{c}-\mathbf{d})=\mathbf{c}-\mathbf{a}$$

上式显示了包含两个向量的独立方程。实际上，上式对 x，y 坐标皆成立，因而包含两个方程。也就是说，可将向量方程调整为两个联立方程，如下所示：

$$t(b_1-a_1)+s(c_1-d_1)=c_1-a_1$$

$$t(b_2-a_2)+s(c_2-d_2)=c_2-a_2$$

其中，a_1，b_1，c_1，d_1 表示为向量 **a**，**b**，**c**，**d** 的 x 分量，即点 A，B，C，D 的 x 坐标。同理，a_2，b_2，c_2，d_2 也存在类似的情况。

【提示】此处应避免将多种概念混为一谈，并区分点、向量以及分量之间的差别。当问题趋于复杂时，这将有助于读者对问题的理解。

1. 向量的实现代码

联立方程可采用与前述方法相同的处理方案进行求解，即向 t 和 s 赋予相关值，此处仅需要使用其中的一个值即可求解当前问题。intersectionPoint()函数采用了一种简化方案：接收 4 个向量参数并返回直线的交点，如下所示：

```
function intersectionPoint(a, b, c, d)
  set tc1 to b[1]-a[1]
  set tc2 to b[2]-a[2]
  set sc1 to c[1]-d[1]
  set sc2 to c[2]-s[2]
  set con1 to c[1]-a[1]
  set con2 to c[2]-a[2]
  set det to (tc2*sc1-tc1*sc2)
  if det=0 then return "no unique solution"
  set con to tc2*con1-tc1*con2
  set s to con/det
  return c+s*(d-c)
end function
```

【提示】在 intersectionPoint()函数中，变量 det 用于(tc2*sc1−tc1*sc2)值计算，即判别式，稍后将对此加以讨论。

2. 返回 t 值

当然，读者还可采用不同方式处理 A 和 C 的位置向量，以及向量 $\overrightarrow{AB}$ 和 $\overrightarrow{CD}$，如下所示：

```
function intersectionTime(p1, v1, p2, v2)
  set tc1 to v1[1]
  set tc2 to v1[2]
  set sc1 to v2[1]
  set sc2 to v2[2]
  set con1 to p2[1]-p1[1]
  set con2 to p2[2]-p1[2]
  set det to (tc2*sc1-tc1*sc2)
  if det=0 then return "no unique solution"
  set con to sc1*con2-sc2*con1
  set t to con/det
  return t
end function
```

intersectionTime()返回 t 值而非交点，如图 5.12 所示，t 和 s 的作用不仅限于确定 P 的位置，其中还包括 P 与点 A，B，C，D 之间的关系。例如，若 t 值为 0.5，则 P 等于 $\mathbf{a} + 0.5\,(\mathbf{b} - \mathbf{a})$，即沿直线 AB 的中途位置。

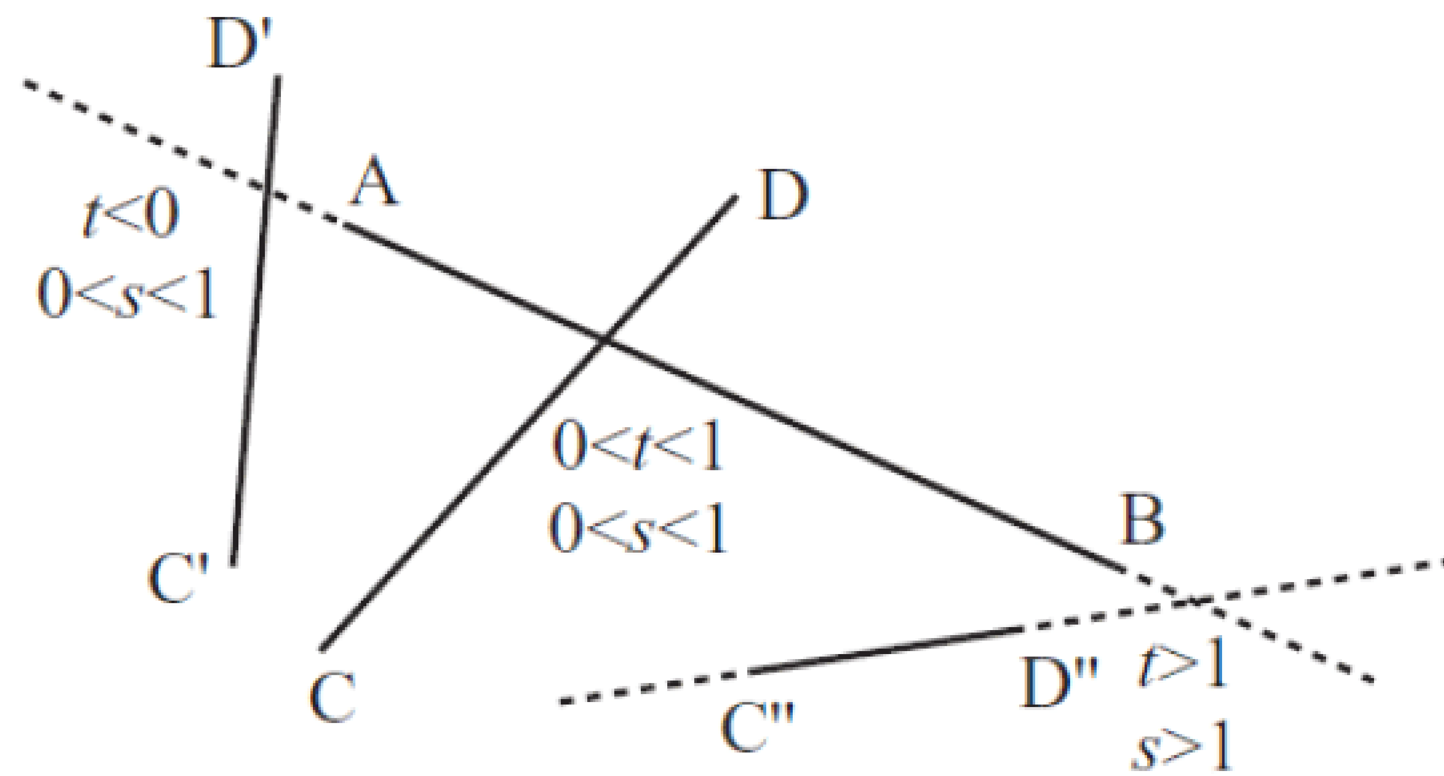


图 5.12　直线交点和向量的参数化结果

总体而言，若 t 位于 0 和 1 之间，则 P 位于点 A 和 B 之间（0 表示点 A，1 表示点 B）；若 t 大于 1，则 P 位于点 B 之外；若 t 小于 0，则 P 位于点 A 之外。

类似地，s 用于确定 P 在直线 CD 上的距离。若 s 和 t 皆位于[0,1]内，则点 P 位于线段 AB 和 CD 的交点处。在其他情况下，P 位于直线至无穷远的投影上。

3. 交线

除了 intersectionPoint()函数和 intersectionTime()函数之外，读者还可尝试编写不同类型的函数，例如，计算两个线段之间的交点。除了返回 t 值，该函数还可用于碰撞检测计算中，如下所示：

```
function intersection(a, b, c, d)
  set tc1 to b[1]-a[1]
  set tc2 to b[2]-a[2]
  set sc1 to c[1]-d[1]
  set sc2 to c[2]-s[2]
  set con1 to c[1]-a[1]
  set con2 to c[2]-a[2]
```

```
  set det to (tc2*sc1-tc1*sc2)
  if det=0 then return "no unique solution"
  set con to tc2*con1-tc1*con2
  set s to con/det
  if s<0 or s>1 then return false
  if tc1<>0 then set t to (con1-s*sc1)/tc1
  otherwise set t to (con2-s*sc2)/tc2
  if t<0 or t>1 then return "none"
  return t
end function
```

4. **注意事项**

在上述 3 个函数中，det 值可能为 0，此时，直线 AB 和 CD 处于平行状态，此时，至少还存在下列两种情况：

- 若线段 AB 和 CD 位于同一条直线上（即 A，B，C，D 共线），则两线段存在公共相交部分，抑或二者彼此分离。
- 若两条线段未处于同一直线上，则二者不相交。

上述两种情形的区分过程并不复杂，并可使用另一个向量方程。若 A，B，C，D 共线，则任意一个顶点均可通过其他两个顶点加以描述。例如，针对某一 k 值，$\mathbf{c} = \mathbf{a} + k(\mathbf{b} - \mathbf{a})$。若该方程中的 k 值位于 0～1 之间，则 C 位于 A 和 B 之间。参数 1 和点 D 也存在着类似的情形。只要 C 或 D 位于线段内，则直线间处于相交状态。

5.5　矩　　阵

类似于向量，矩阵也采用数学阵列表示列表信息。本小节讨论矩阵以及矩阵算术的基本知识。

5.5.1　矩阵基础知识

为了进一步理解矩阵和向量之间的差别，首先可将向量视为一维阵列，即分量列表，并以此表示固定数量方向上的运动。即使在三维环境中，向量依然表示为一维对象，例如$\begin{pmatrix}1\\2\\3\end{pmatrix}$，且针对各方向均包含单一值。

相比较而言，矩阵包含两个方向上的数据值，并采用表格式予以显示，例如$\begin{pmatrix}1 & 3\\-2 & 2\end{pmatrix}$。其中，水平方向上的数据称作行，垂直方向上的数据称作列。因此，读者可将矩阵视为行、列关联的数据表。表 5.1 显示了电子器件的价格。

表 5.1　电子器件价格表

商　　品	小	中	大
Widgets	1.20美元	3.00美元	4.00美元
Gizmos	10.00美元	15.00美元	20.00美元
Whatsits	5.25美元	8.50美元	11.00美元

若考察 Gizmos 行并定位至“大”一列，对应商品的价格为 20.00 美元。

矩阵采用行、列方式加以定义，若矩阵出现于方程中，则应采用简写方式对其加以书写。通常情况下，矩阵多采用黑体字母表示（例如 **G**）。下列内容显示了矩阵的赋值方式：

$$\mathbf{G}=\begin{pmatrix}1.2 & 3 & 4\\ 10 & 15 & 20\\ 5.25 & 8.5 & 11\end{pmatrix}$$

这里，可采用“×”号关联矩阵的行和列。若 **G** 包含 3 行、3 列，则该矩阵称作 3×3 矩阵。若加入第 4 列数据，则矩阵表示为 3×4 矩阵。另外，若矩阵包含相同的行数和列数，则该矩阵称作方阵。根据矩阵的定义，读者还可定义转置矩阵，即矩阵的行、列互换并采用上标 T 表示。下列内容显示了矩阵 **G** 的转置矩阵：

$$\mathbf{G}^{\mathrm{T}}=\begin{pmatrix}1.2 & 10 & 5.25\\ 3 & 15 & 8.5\\ 4 & 20 & 11\end{pmatrix}$$

对于 $n\times m$ 矩阵的转置矩阵，其尺寸为 $m\times n$。需要注意的是，转置矩阵并未改变左上方至右下方对角线上的内容，该对角线称作矩阵的主对角线。

由于向量可表示为 $n\times 1$ 矩阵，因而可使用转置符号将其表示为行矩阵。此时，$n\times 1$ 矩阵变为 $1\times n$ 矩阵。也就是说，矩阵 $\begin{pmatrix}1\\2\\3\end{pmatrix}$ 表示为 $(1\ 2\ 3)^{\mathrm{T}}$。除了圆括号之外，还可采用尖角括号表示矩阵，即 $\langle 1,2,3\rangle$。

在程序设计中，矩阵常通过数组的数组表示。对于 3×3 矩阵，一个数组用于表示行数据，且该数组包含 3 个数组，各数组分别包含 3 个元素，每一个元素表示为列中的数值。因此，矩阵 **G** 可定义为[[1.2, 10, 5.25],[3, 15, 8.4],[4.5, 20, 11]]这一数组形式。

5.5.2　行列式

方阵包含一类称作矩阵行列式的关联值，类似于向量的长度，行列式可视为矩阵的基本特征。行列式往往采用竖线表示，即矩阵 **M** 的行列式记为|**M**|，对应函数记为 det(**M**)。

矩阵的行列式等同于其转置矩阵的行列式。针对 2×2 矩阵 $\begin{pmatrix}a & b\\ c & d\end{pmatrix}$，其行列式等于 $ad-bc$。对于较大的矩阵，行列式也随之变得越发复杂，读者可尝试采用递归函数对其求解，该函数接收

一个方阵作为参数。对此，determinant()函数提供了一类解决方案，如下所示：

```
function determinant(m)
  set size to the number of elements in m
  if size=1 then return m[1][1]
  set mult to 1
  set sum to 0
  repeat for i=1 to size
    set el to m[1][i]
    set newmatrix to an empty array
    repeat for j=2 to size
      append m[j] to newmatrix
      remove the I'th element of this row
    end repeat
    add el*mult*determinant(newmatrix) to sum
    multiply mult by -1
  end repeat
  return sum
end function
```

【提示】 矩阵、向量和标量均可视为数学阵列通用类 tensors 的特例，该类常用于描述数学空间区域的数值变化。当与张量协同工作时，往往意味着更多的数学技巧。张量是物理学中不可或缺的内容之一，例如广义相对论或电磁学。另外，旋转行为中也常可看到其身影，例如惯性张量。

5.5.3　矩阵算术

类似于向量算术，矩阵算术始于标量和矩阵之间的操作，随后逐渐过渡到矩阵间的运算。针对矩阵与标量的乘法运算，可在矩阵中的各项数据与标量之间执行乘法运算，如下所示：

$$2\times\begin{pmatrix}1 & 3\\4 & 5\end{pmatrix}=\begin{pmatrix}2 & 6\\8 & 10\end{pmatrix}$$

矩阵间的加法可视为最为简单的矩阵操作，其中，两个同尺寸的矩阵其对应数据项之间进行加法运算，如下所示：

$$\begin{pmatrix}1 & 3\\4 & 5\end{pmatrix}+\begin{pmatrix}2 & -1\\0 & 3\end{pmatrix}=\begin{pmatrix}3 & 2\\4 & 8\end{pmatrix}$$

此类操作易于理解，然而，当矩阵间执行乘法操作时，情况则有所不同。当然，若数据项相对简单，对应处理过程并不复杂。例如，$l\times n$ 矩阵 **L** 与 $n\times m$ 矩阵 **M** 之间的乘法操作。

对此，可将 **L** 第一行中的首个元素乘以 **M** 第一列中的首个元素，并对两个矩阵中第一行和第一列中的其他元素执行相同的计算。随后，可将求和结果置于新矩阵 **N** 中的左上角。

针对矩阵 **L** 的 i 行和矩阵 **M** 的 j 列，重复执行上述过程，进而得到矩阵 **N** 的第 i 行和第 j 列数据值。需要注意的是，仅当矩阵 **L** 的列数与矩阵 **M** 的行数相等时，上述计算方为有效。

matrixMultiply()函数接收两个参数 l 和 m，即执行乘法操作的两个矩阵。随后，该函数返回

两个矩阵相乘后的新矩阵，如下所示：

```
function matrixMultiply(l, m)
  set n to a blank array
  repeat with i=1 to the number of rows of l
    set r to a blank array
    repeat with j=1 to the number of columns of m
      set sum to 0
      repeat with k=1 to the number of columns of l
        add l[i][k]*m[j][k] to sum
      end repeat
      append sum to r
    end repeat
    append r to n
  end repeat
  return n
end function
```

通过实践可知，矩阵乘法并不满足交换律。也就是说，$\mathbf{LM} \neq \mathbf{ML}$。实际上，矩阵的逆序乘法操作并不具有实际意义。即使对于方阵，矩阵间乘法运算的结果也随着计算顺序的不同而产生变化。另外一方面，转置矩阵的计算过程也将有所不同。例如，若 $\mathbf{LM} = \mathbf{N}$，则有 $\mathbf{M}^{\mathrm{T}}\mathbf{L}^{\mathrm{T}} = \mathbf{N}^{\mathrm{T}}$。

【提示】需要注意的是，$\mathbf{u}^{\mathrm{T}}\mathbf{v}$ 的计算结果将生成 $\mathbf{u}$ 和 $\mathbf{v}$ 的点积（或 1×1 矩阵，其唯一数据项为点积结果）。

虽然矩阵乘法不满足交换律，但该操作却满足结合律，即 $\mathbf{L}(\mathbf{MN})=(\mathbf{LM})\mathbf{N}$，以及针对加法的分配律，即 $\mathbf{L}(\mathbf{M}+\mathbf{N})=\mathbf{LM}+\mathbf{LN}$。对于方阵，矩阵乘法的行列式还满足$|\mathbf{LM}|=|\mathbf{L}|\,|\mathbf{M}|$这一条件。

针对各种尺寸的方阵，均存在单位矩阵 $\mathbf{I}$，其乘法运算使得另一矩阵保持不变，即 $\mathbf{IM}=\mathbf{MI}=\mathbf{M}$。另外，针对各乘法运算，单位矩阵也需要选取适当的尺寸。在主对角线中，单位矩阵的数据值均为 1，而其他位置则为 0。例如，2×2 单位矩阵表示为$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$。

针对任意行列式为非 0 的方阵 $\mathbf{M}$，存在唯一的逆矩阵 $\mathbf{M}^{-1}$，并满足 $\mathbf{MM}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$。对于 2×2 矩阵$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$，其逆矩阵等于$\dfrac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$，这也构成了联立方程的另外一种求解方法。对此，可将一组联立方程“编码”为一个矩阵，假设方程组如下所示：

$$ax + by = p$$
$$cx + dy = q$$

其等价形式如下所示：

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} p \\ q \end{pmatrix}$$

其中，$(x \quad y)^{\mathrm{T}}$ 表示为独立的“二维”未知项。

随后，方程两侧左乘逆矩阵，如下所示：

$$\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} x \\ y \end{pmatrix}=\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix}$$

【提示】此处应注意左乘表达式。由于矩阵乘法不满足交换律，因而左乘和右乘视为不同的操作。

当前，可通过单一矩阵计算 x 和 y 值，如下所示：

$$\begin{pmatrix} x \\ y \end{pmatrix}=\frac{1}{ad-bc}\begin{pmatrix} dp-pq \\ aq-cp \end{pmatrix}$$

读者可采用与联立方程类似的求解方式计算大于 2×2 矩阵的逆矩阵，针对矩阵行执行线性操作（执行标量乘法并添加线性组合），进而将其转换为单位矩阵。若针对原始单位矩阵执行相同操作，则结果将转换为原始矩阵的逆矩阵。

5.5.4　基于转换的矩阵

类似于向量（按照某一方向移动），矩阵也可视为一类指令。矩阵和向量的乘法运算结果将得到一个新向量，因而矩阵体现了向量的解释方式。实际上，矩阵可视为一类空间转换操作，下列内容显示了二维空间内的矩阵转换示例：

- 对于矩阵$\begin{pmatrix} n & 0 \\ 0 & n \end{pmatrix}=n\mathbf{I}$，若将其乘以任意向量 $\mathbf{v}$，则对应结果等于该向量与 n 之间的计算结果，即$(n\mathbf{I})\mathbf{v}=n(\mathbf{I}\mathbf{v})=n\mathbf{v}$。该向量体现了一类缩放操作。任何包含非 1（或 0）行列式的矩阵，均包含了相应的缩放因素。
- 若矩阵$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$与向量左乘，则向量的 x 坐标反转，也就是说，向量相对于 y 轴反射。相应地，包含负行列式的矩阵均可视为一类反射操作，并可能存在缩放行为。
- 若矩阵$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$与向量左乘，则可得到 x，y 互换的结果向量。另外，x 坐标反转并生成一个法线向量，即围绕原点顺时针旋转 90°。
- 若矩阵$\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$与向量相乘，则结果向量可描述为：围绕原点顺时针旋转 θ。需要注意的是，由于 $\cos^2(\theta)+\sin^2(\theta)=1$，因而矩阵的行列式为 1。相应地，该向量未经缩放或反射。
- 若矩阵$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$与向量相乘，则对应向量斜向平行于 x 轴（倾斜量与 y 分量呈比例）。该矩阵体现了一类剪切行为，且行列式为 1。

需要说明的是，并非所有转换均可通过矩阵这一方式表达。特别地，平移操作可通过加入一个固定向量得以实现，而非矩阵的乘法运算。基于原点的各转换操作均可采用矩阵加以表示，读者可尝试使用矩阵的乘法运算规则计算多个转换的组合结果。例如，若围绕原点将一个三角形顺时针旋转 75°，将其尺寸放大两倍并针对 x 轴执行反射操作，相关步骤如下所示：

- 三角形各顶点的位置向量乘以矩阵 $\mathbf{R}=\begin{pmatrix}\cos(75^\circ) & \sin(75^\circ)\\ -\sin(75^\circ) & \cos(75^\circ)\end{pmatrix}$。
- 结果向量乘以矩阵 $\mathbf{S}=\begin{pmatrix}2 & 0\\ 0 & 2\end{pmatrix}$。
- 结果向量乘以矩阵 $\mathbf{T}=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$。

这也意味着，若某一向量包含位置向量 **a**，其最终结果可表示为 $\mathbf{T}(\mathbf{S}(\mathbf{Ra}))=(\mathbf{TSR})\mathbf{a}$。若将该过程整合为单一步骤，则需要首先计算 **TSR** 矩阵，并将该矩阵应用于三角形的全部顶点上。

根据矩阵乘法可知，转换操作不符合交换律。若采用不同顺序执行上述 3 项操作，将得到不同的计算结果。对此，假设读者左转并朝向一面镜子，相应地，若先期朝向镜面并于随后左转，则二者结果显然不同。

在结束本章内容之前，下面快速考察基向量 $\mathbf{i}=(1\ 0)^{\mathrm{T}}$ 和 $\mathbf{j}=(0\ 1)^{\mathrm{T}}$ 在 $\mathbf{M}=\begin{pmatrix}a & b\\ c & d\end{pmatrix}$ 转换作用下的操作结果。不难发现，$\mathbf{Mi}=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix}=\begin{pmatrix}a\\ c\end{pmatrix}$ 且 $\mathbf{Mj}=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix}=\begin{pmatrix}b\\ d\end{pmatrix}$，矩阵 **M** 的各列将生成基向量的转换结果。若基向量的转换结果已知，则可从中获取各种有用信息，并可通过该结果计算转换矩阵。

矩阵的一个较为重要的属性是特征向量及其关联的特征值，这可视为标定矩阵的一类简单方案。实际上，特征向量体现了基于特定转换且与自身相关的倍向量，对应倍数称作该向量的特征值。若 **p** 表示为 **M** 的特征向量，λ 表示为对应的特征值，则有如下等式：

$$\mathbf{Mp}=\lambda\mathbf{p}$$

【提示】 严格地讲，读者应区分左特征向量和右特征向量。左特征向量类似于前述内容讨论的特征向量，并在矩阵左侧执行乘法运算；当在矩阵右侧执行乘法运算时，右特征向量则更为常见。如无特殊说明，此类特征向量多指右特征向量。尽管左、右特征向量彼此不同，但针对既定矩阵，二者的特征值则保持一致。

5.6 本 章 练 习

【练习 5.1】试编写一组函数，例如 drawArrowhead(linesegment,size,angle)和 drawKite(linesegment, height, width)函数，并根据初始简单参数生成复杂的形状。

除了上述两种形状之外，读者还可尝试前述内容所讨论的字母形状。对此，可根据宽度、高度和角度创建可变字体。同时，读者还可生成简单的 3D 效果，即基于点集的斜角效果。

【练习 5.2】 根据本章示例，尝试编写 calculateTrajectory(oldPosition, newPosition, speed)函数，并预计算速度向量以及其他必要的运动参数。

该函数应包含与本章 calculateTrajectory()函数相同的参数集。

5.7　本 章 小 结

本章讨论了向量、矩阵的算术和代数运算，并回顾了与空间相关的基本工具。其中，对应内容介绍了如何使用向量描述位置和运动，并执行较为复杂的计算，例如直线相交。除此之外，本章还探讨了矩阵如何对空间执行转换操作以及基向量等内容。

第 6 章将继续考察代数运算，并对相关函数加以进一步分析。

至此，读者应掌握如下内容：

- 向量的含义、基于分量的向量描述方式以及如何执行基本的算术运算，例如向量加法、向量减法以及向量的缩放操作。
- 向量的大小、范数和法线向量及其基于既定向量的计算方式。
- 如何计算两个向量的标量（点）积，以及标量积的具体含义。
- 如何使用组合向量描述空间内的已知位置。
- 向量的参数化描述方式，以及如何表达“直线 AB 上的向量”这一类概念。
- 如何通过几何方式以及点积方式计算各方向上的向量分量(将其置于不同的基向量上)。
- 采用联立方程组求解向量方程，进而计算两条直线的交点。
- 矩阵的含义以及如何执行矩阵计算。
- 如何根据矩阵求解联立方程。
- 通过矩阵执行空间的多次转换操作。

第 6 章　微　积　分

本章包含如下内容：

- 概述。
- 微分和积分。
- 微分方程。
- 近似方案。

6.1　概　　述

作为第一部分内容的最后一章，本章讨论与极限相关的数学知识，即微积分。尽管程序设计中仅涉及有限微积分知识，且多与物理问题相关，但这一部分内容对于读者而言依然是十分必要的。本章力争做到有的放矢，并为后续章节打下坚实的基础。

6.2　微分和积分

第 3 章和第 4 章曾分别讲述了微积分方面的内容，即函数图和梯度计算，微积分与函数图和梯度之间存在多种应用方式。

6.2.1　函数梯度

前述直线斜率计算曾涉及梯度问题，例如直线 $y = 2x + 1$。然而，梯度这一概念仍过于笼统。在图 6.1 中，假设存在一条连续、平滑曲线且点 P_0 位于该曲线上，并在其中选取一系列的数据点 P_1，P_2，…，且每一个点均比前一个点更接近于点 P_0。当绘制连接 P_0 和各点的直线时，对应结果逐渐趋向于一条特定直线，即曲线上位于 P_0 点的切线，该切线的梯度称作点 P_0 处的曲线梯度。

当讨论极限这一概念时，常会使用到希腊字母 δ（delta），相应地，可采用图 6.1 中的方法计算特定 x 值处的函数梯度，例如 x_0。在图 6.2 中，若使用较小值 δ 并计算 $f(x_0 + \delta)$ 值，随着 δ 值逐渐减小，将生成一系列的曲线点，并依次接近于 $(x_0, f(x_0))$。

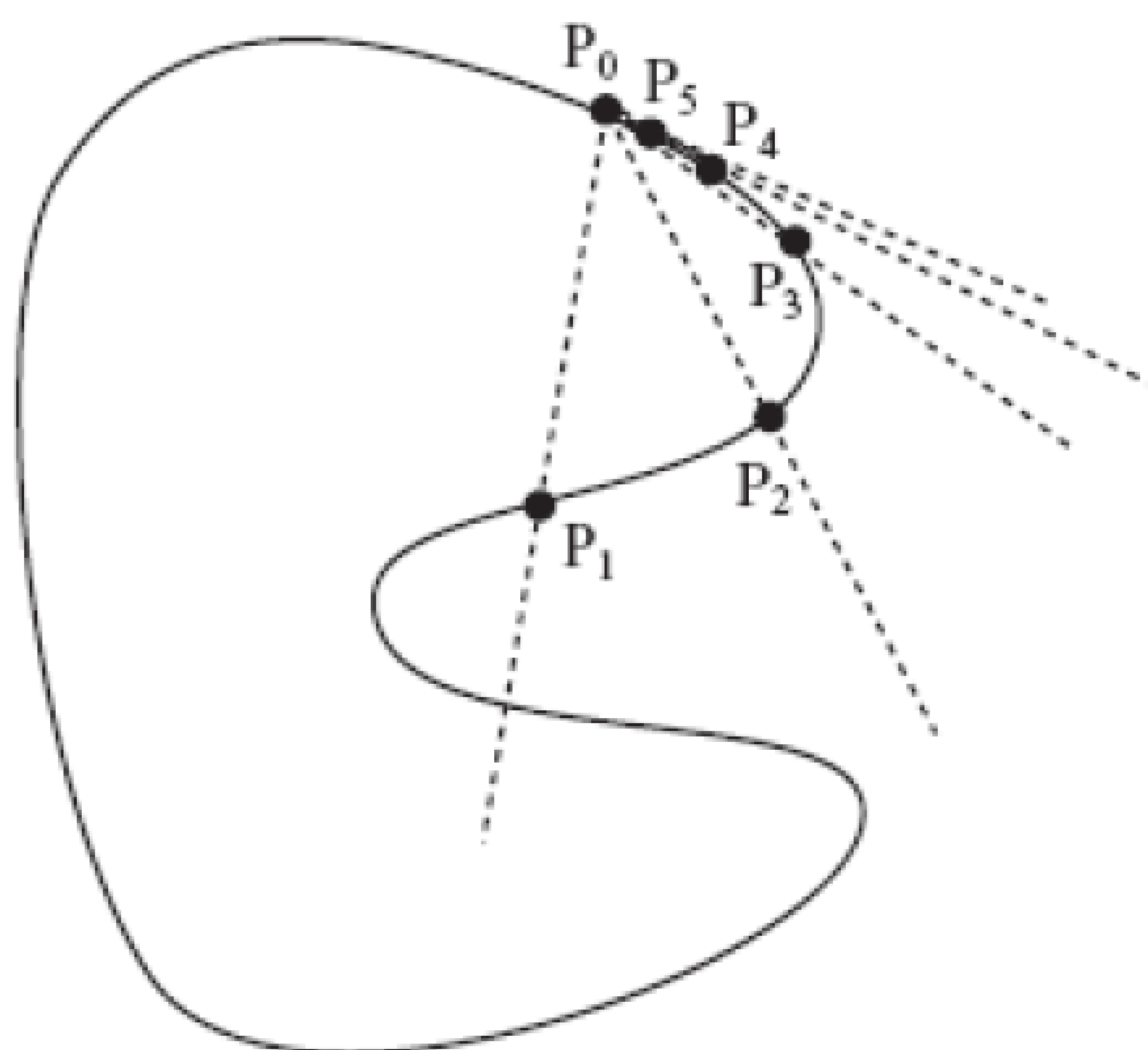


图 6.1　特定点处的曲线梯度计算

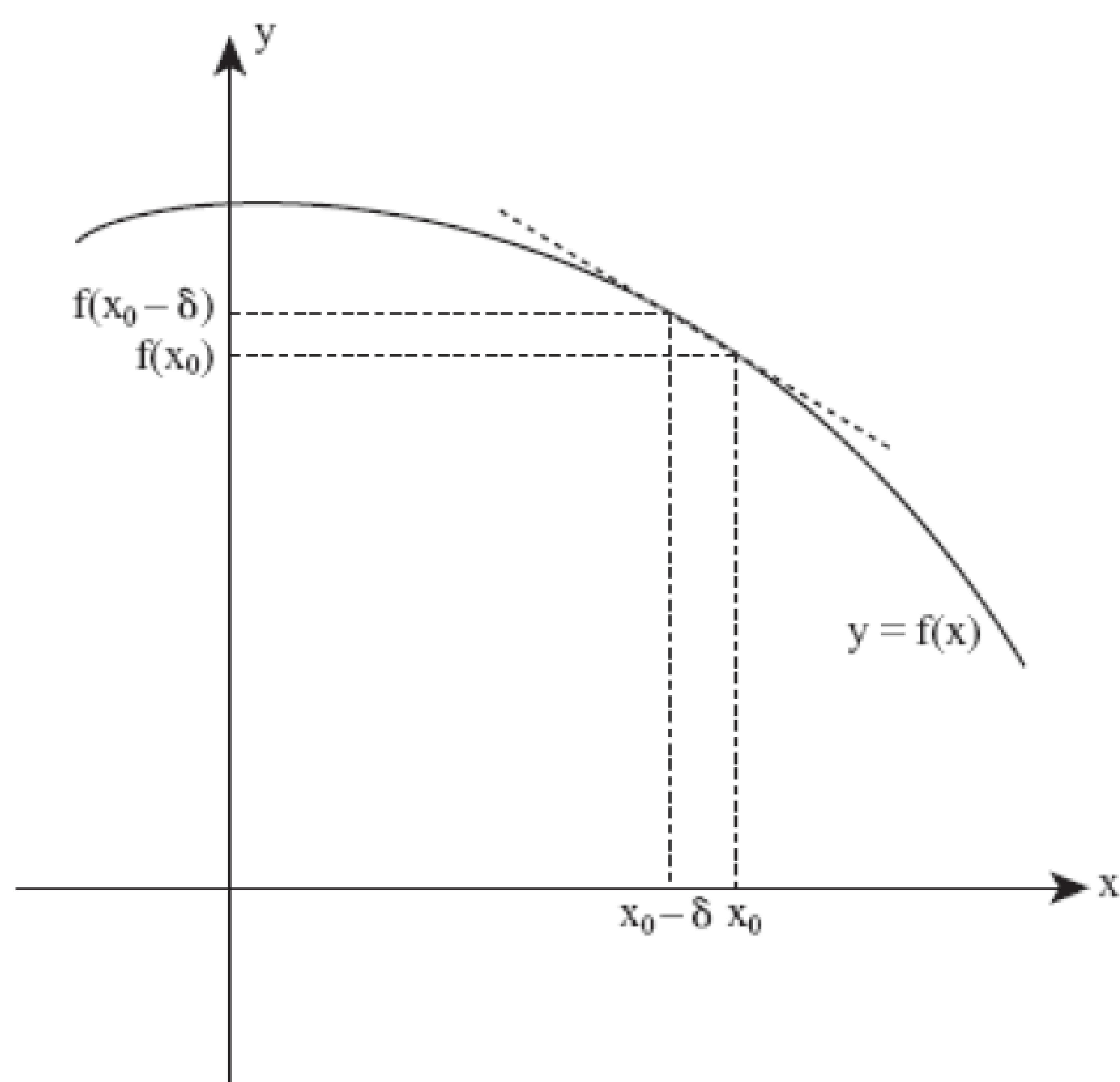


图 6.2　函数梯度

如前所述，连接某一点与目标点之间的梯度可通过二者间垂直距离除以水平距离得到，对应梯度值如下所示：

$$\frac{f(x_0+\delta)-f(x_0)}{\delta}$$

这意味着，随着 δ 趋于 0，函数梯度等于其极限值，如下所示：

$$\lim_{\delta\to\infty}\frac{f(x_0+\delta)-f(x_0)}{\delta}$$

为了对此予以进一步说明，下面对抛物线函数 ax^2+bx+c 加以考察，并计算 x_0 处的切线，如下所示：

$$\begin{aligned}f(x_0+\delta)&=a(x_0+\delta)^2+b(x_0+\delta)+c\\&=ax_0^2+2ax_0\delta+a\delta^2+bx_0+b\delta+c\\&=(2ax_0+b)\delta+a\delta^2+ax_0^2+bx_0+c\end{aligned}$$

且有

$$f(x_0)=ax_0^2+bx_0+c$$

当计算梯度时，需要计算下列值：

$$\begin{aligned}\lim_{\delta\to 0}\frac{f(x_0+\delta)=f(x_0)}{\delta}&=\lim_{\delta\to 0}\frac{((2ax_0+b)\delta+a\delta^2+ax_0^2+bx_0+c)-(ax_0^2+bx_0+c)}{\delta}\\&=\lim_{\delta\to 0}\frac{(2ax_0+b)\delta+a\delta^2}{\delta}\\&=\lim_{\delta\to 0}2ax_0+b+a\delta\end{aligned}$$

当 δ 趋于 0 时，上述表达式将逐渐接近于 $2ax_0+b$。据此，$x=1$ 处的梯度值为 $2a+b$。

6.2.2 微分计算

6.2.1 节中的处理过程称作函数的微分计算，对应示例使用了变量 x_0。需要说明的是，x_0 仅表示为一个变量，读者可根据个人喜好对其加以定义。微分计算始于 $f(x)$，并生成新的函数 $g(x)$。新函数体现了函数 f 于 x 各值处的梯度。通常情况下，$g(x)$ 可记为 $\frac{df}{dx}$ 或 $f'(x)$，即函数 f 的导数。

其中，D 表示“较小的变化”，因而表达式 $\frac{df}{dx}$ 表示基于较小 x 值变化的 $f(x)$ 变化量。也就是说，导数体现了相对于变量 x 的函数变化率；换而言之，导数表示 x 值变化时 $f(x)$ 值的变化速度。

二阶导数可视为对一阶导数求导执行微分计算，这将生成新函数 $\frac{d^2f}{dx^2}$。同时，该结果还可记为 $f''(x)$，并表示函数 f 梯度的变化率。针对其他高阶导数，其运算符号亦保持类似的模式。

除了上述显示的导数符号之外，还存在另一类较为常见的标识方案，该方案多出现于物理函数中，对应主变量表示为时间值，或者包含参数 t 的参数方程。其中，基于时间的导数采用“.”表示。例如，针对函数 $y(t)$，一阶导数表示为 $\dot{y}(t)$ 或 $\dot{y}$，二阶导数表示为 $\ddot{y}(t)$ 或 $\ddot{y}$。

该过程称作数值微分，据此，读者可对大多数连续函数执行微分计算。下列内容显示了一些常见的计算规则。

（1）若 a 为常量，则有：

$$\frac{d}{dx}(af(x))=a\frac{d}{dx}(f(x))$$

（2）若 f 和 g 表示为函数，则有：

$$\frac{d}{dx}(f'(x)+g(x))=f'(x)+g'(x)$$

（3）若 f 和 g 表示为函数，则有：

$$\frac{d}{dx}(f(x)g(x)) = f'(x)g(x) + g'(x)f(x)$$

（4）作为链式法则，若 f 和 g 表示为函数，则有：

$$\frac{d}{dx}(f(g(x))) = f'(g(x))g'(x)$$

（5）常量的导数为 0。

（6）x 的导数为 1。

6.2.3　应用示例

链式法则体现了微分计算的一个重要特征，在大多数场合下，dx 和 dy 与常规变量并无太多差异。例如，当处理分数时，同样可执行消去操作。链式法则表明，消去 dg 则可得到等式 $\frac{df}{dg}\times\frac{dg}{dx}=\frac{df}{dx}$。

若将链式法则应用于多项式上，例如 ax^3+bx^2+cx+d，其功效则十分明显。通过上述规则（2），可分别处理各个数据项；通过规则（1），则可忽略系数项。针对某一整数 n，计算可始于 $f(x)=x^n$ 的微分计算。通过规则（3），则有如下算式：

$$\frac{d}{dx}(x^n)=\frac{d}{dx}(x^{n-1}\times x)$$

根据规则（5），可知 $\frac{d}{dx}(x)=1$，因而有：

$$\begin{aligned}\frac{d}{dx}(x^n) &= \frac{d}{dx}(x^{n-1})\times x + x^{n-1}\\ &= \frac{d}{dx}(x^{n-2})\times x^2 + x^{n-2}\times x + x^{n-1}\\ &= \cdots\\ &= \frac{d}{dx}(x)\times x^{n-1} + x^{n-1} + \cdots + x^{n-1}\\ &= nx^{n-1}\end{aligned}$$

读者可尝试使用较小值 4，对应结果为 $\frac{d}{dx}(x^4)=4x^3$。

通过该结果以及规则（1）、（2）、（5）和（6），则可得到如下算式：

$$\frac{d}{dx}(ax^3+bx^2+cx+d) = 3ax^2+2bx+c$$

总体而言，阶数为 n 的多项式导数可得到阶数为 $n-1$ 的多项式。

又如，若对 $y=(3x+2)^2$ 执行微分计算，则同样可使用链式法则。假设 $g=3x+2$ 且 $y=g^2$，则有：

$$\frac{dy}{dx}=\frac{dy}{dg}\times\frac{dg}{dx}=2g\times 3=6(3x+2)$$

读者可通过下列方式检测计算结果，即消除 y 表达式中的括号，并对多项式直接进行微分计算。

6.2.4 导数信息

如图 6.3 所示，函数的导数可返回某些重要信息，其中包括以下内容：

- 针对值 x，若函数的导数为 0，则该函数在 x 处具有转折点。这意味着，连续函数在一阶导数值处存在最大值或最小值，反之则不成立，即各转折点不一定是最大值或最小值，也可能为其他折点。
- 若函数在特定 x 值处包含渐近线，则其导数在同一值处同样包含一条渐近线。
- 若函数的导数为负值，则该函数自左至右倾斜；若为正值，则函数自右至左倾斜。

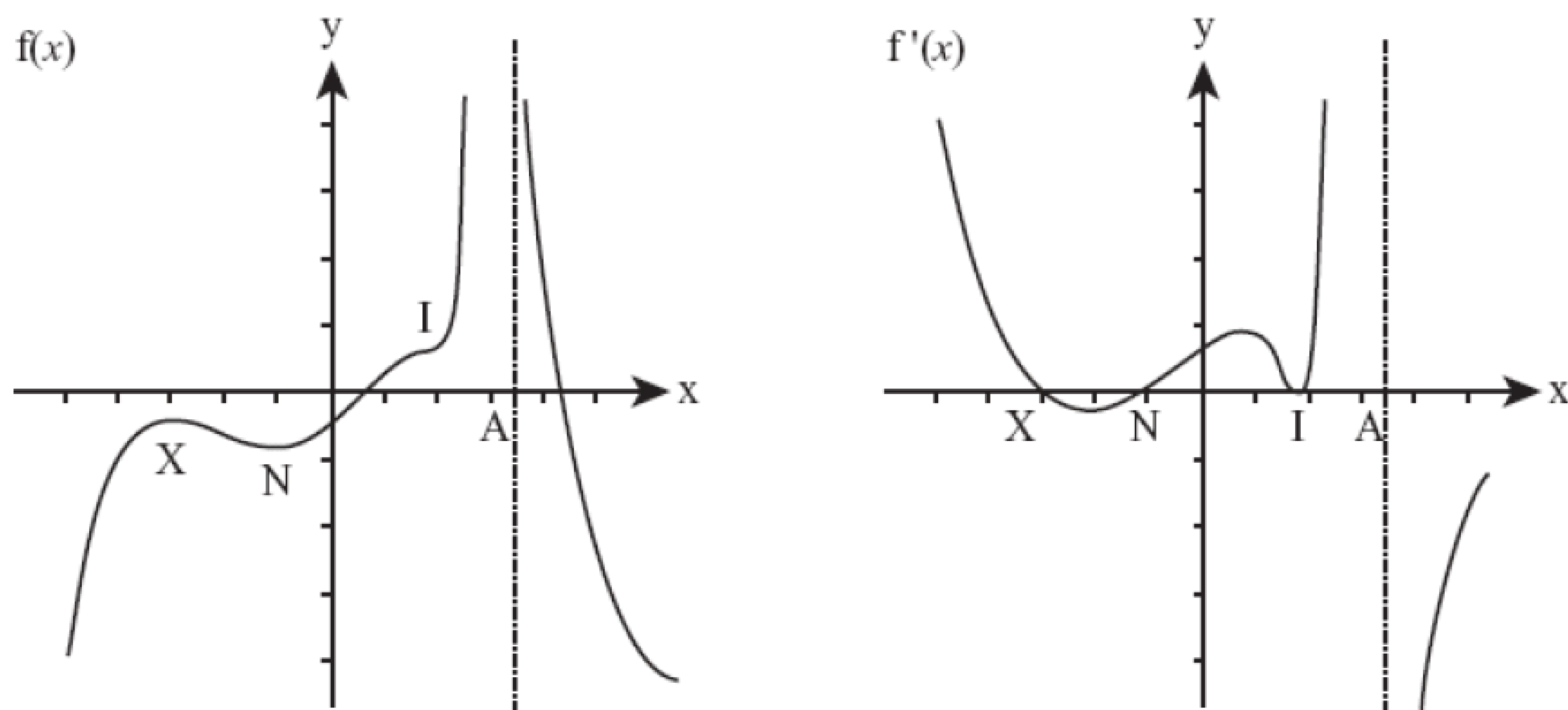


图 6.3 函数及其一阶导数

在图 6.3 中，标记为 N、X 以及 I 的点分别表示最小值、最大值以及 $f(x)$ 的折点。在当前上下文环境下，术语“最大值”和 “最小值”定义为局部最大值和局部最小值，而非全局最大值和全局最小值。由于当前函数并不存在全局最大值，因而于 A 处包含一条渐近线。

当考察图 6.3 中的函数图时，不难发现各折点处的导数为 0。最大值和最小值对应的点其导数穿越 x 轴；若导数与 x 轴相切，则此类数据点为折点。

若执行二阶导数计算，则可获取折点类型。其中，正二阶导数表示最小值，而负二阶导数表示为最大值；若二阶导数为 0，则对应点为折点（拐点）。

【提示】 严格地讲，折点无须包含 0 值一阶导数，并可表示为任一点，其一阶导数包含最大值或最小值。此处，二阶导数为 0。

6.2.5 对数和指数的微分运算

多项式并非仅是可微的函数，许多常见的函数也包含简单的导数计算，例如 exp()和 log()函数。指数函数 exp()较为常见，且常出现于微积分运算中。回忆一下，曾在第 2 章讨论到，自然对数的底数 e 等于 $\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots$。总体而言，针对于下列算式：

$$e^x=\frac{1}{0!}+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots$$

若采用 1 替换式中的 x 项，即可得到原始的 e 值。上式的证明过程具有一定的技巧性，读者可通过相关特例对其进行归纳，例如 x=2。

上式中各项均表示为 $\frac{nx^{n-1}}{n!}$。针对分数定义，$\frac{n}{n!}=\frac{1}{(n-1)!}$，因而上式衍变为下列形式：

$$0+\frac{1}{0!}+\frac{x}{1!}+\frac{x^2}{2!}+\cdots$$

需要注意的是，e^x 函数稍显独特，即 e^x 函数的导数为自身。

对数则具有如下特征：

$$\frac{d}{dx}\log_e(x)=\frac{1}{x}$$

回忆一下，对于包含一条渐近线的函数，其渐近线位置与导数的位置相同。换而言之，对数函数以及 $\frac{1}{x}$ 函数均在 $x=0$ 处包含一条渐近线。

6.2.6　三角函数的微分运算

e 值与三角函数之间具有紧密的联系，二者间的微分运算也具有相近的特征。下列内容表示为 sin(x)和 cos(x)的无穷级数：

$$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots$$

$$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\cdots$$

若对上述无穷级数执行微分运算，则有 $\frac{d}{dx}\sin(x)=\cos(x)$且 $\frac{d}{dx}\cos(x)=-\sin(x)$。这也意味着，两个函数等于其二阶导数的相反数，即 $\frac{d^2}{dx^2}\sin(x)=-\sin(x)$且 $\frac{d^2}{dx^2}\cos(x)=-\cos(x)$。

这里，可通过乘积法则和链式法则对 tan(x)执行微分计算，如下所示：

$$\begin{aligned}\frac{d}{dx}\tan(x)&=\frac{d}{dx}\frac{\sin(x)}{cox(x)}\\&=\frac{1}{\cos(x)}\times\frac{d}{dx}\sin(x)+\sin(x)\times\frac{d}{dx}\frac{1}{\cos(x)}\end{aligned}$$

此处，令 g=cos(x)，则有如下算式：

$$\begin{aligned}\frac{d}{dx}\frac{1}{\cos(x)}&=\frac{d}{dg}\frac{1}{g}\times\frac{dg}{dx}\\&=\frac{d}{dg}g^{-1}\times\frac{d}{dx}\\&=(-g^{-2})\times(-\sin(x))\end{aligned}$$

$$=\frac{\sin(x)}{\cos^2(x)}$$

$$=\frac{\tan(x)}{\cos(x)}$$

最终结果为：

$$\begin{aligned}\frac{d}{dx}\tan(x)&=\frac{1}{\cos(x)}\times\frac{d}{dx}\sin(x)+\sin(x)\times\frac{\tan(x)}{\cos(x)}\\&=\frac{1}{\cos(x)}\times\cos(x)+\sin(x)\times\frac{\tan(x)}{\cos(x)}\\&=1+\tan^2(x)\end{aligned}$$

下列内容表明，反三角函数的导数并不包含三角部分：

- $\frac{d}{dx}\tan^{-1}(x)=\frac{1}{1+x^2}$。
- $\frac{d}{dx}\sin^{-1}(x)=\frac{1}{\sqrt{1-x^2}}$。
- $\frac{d}{dx}\cos^{-1}(x)=-\frac{1}{\sqrt{1-x^2}}$。

需要注意的是，平方根意味着，当$|x|>1$时，$\sin^{-1}$和$\cos^{-1}$均不包含定义明确的导数。

6.2.7 参数方程和偏导数

当对形如和 $y(t)$ 与 $x(t)$ 的参数函数计算导数时，下面的示例显示了如何“消除” dx 项。假设计算始于包含参数 t 的特定点，并小幅度变化 t 值。与此对应的是，x 坐标和 y 坐标分别通过 $\frac{dx}{dt}$ 和 $\frac{dy}{dt}$ 初始变化。当计算该点处的曲线梯度时，则有如下算式：

$$\frac{dy}{dt}(t)=\frac{\dot{y}}{\dot{x}}$$

该式表示为 t 的函数——由于 t 值通常为已知内容，因而该参数形式十分有用。在第 3 章中的抛物线示例中（如图 3.6 所示），参数方程表示为 $\dot{x}=2at$（$=y$），$\dot{y}=2a$，因而有 $\frac{dy}{dx}=\frac{2a}{2at}=\frac{1}{t}$。

另一种微分处理方案则集中于多个变量，例如 $z=x^2-2xy+y^2$ 函数，该函数可进一步分解为 $z=(x-y)^2$。这里，读者可尝试将该函数绘制于三维空间表面上，并将 x 和 y 轴定义为水平平面，z 轴则处于垂直状态。

通过观察可知，与单变量函数不同，表面具有一个切面而非切线。其中，平面可通过两种方式加以定义：描述法线的一个 3D 向量，或者两个位于该平面内的 3D 向量。对此，可采用偏微分计算予以实现。此时，针对表面上的任意一点，在 x 和 y 方向上存在两条穿越该点的曲线。当这两条曲线的梯度计算完毕后，即可得到所需的两个向量。

梯度计算可通过表面偏导数予以实现，对应过程与标准导数计算并无太多异处。期间，可将

某一变量视为常量，例如，表面 $z=x^2-2xy+y^2$ 将生成 $\frac{\partial z}{\partial x}=2x-2y$，$\frac{\partial z}{\partial y}=2y-2x$。这里，符号∂表示偏导数。

6.2.8　积分运算

除了微分计算之外，对于函数 g，还存在另一种方式可计算函数 f，其导数 f' 等于函数 g，该过程称作积分运算，函数 f 称作 g 的积分。积分运算通常包含两种形式，不定积分计算相关函数而非某一特定值，定积分则计算某一对应值。

根据前述章节所讨论的内容，若考察函数 f 与函数 g 之间的运动行为，可知函数 f 并不唯一。由于可向函数 f 加入任意常量并保持其导数不变，因而针对参数 c，函数 g 的积分可能是函数 $f(x)+c$ 一族中的任一成员。最终，除了数值 c 之变化外，不定积分保持唯一。

同时，积分还包含另一层含义，即曲线 $g(x)$ 下方的面积，该面积形状位于曲线和 x 轴之间。这里的问题是，$f(x)$ 值如何体现面积值？该面积值体现了何种含义？如同微分计算，积分如何表达极限这一概念？如图 6.4 所示，函数 g 在 x 处的积分值可视为曲线面积中狭长“切片”的面积值。若仔细观察曲线，则会发现基于微分的积分关联方式。其中，位于 x 处的曲线越发陡峭，则该点处曲线下方的切片面积也就越大。

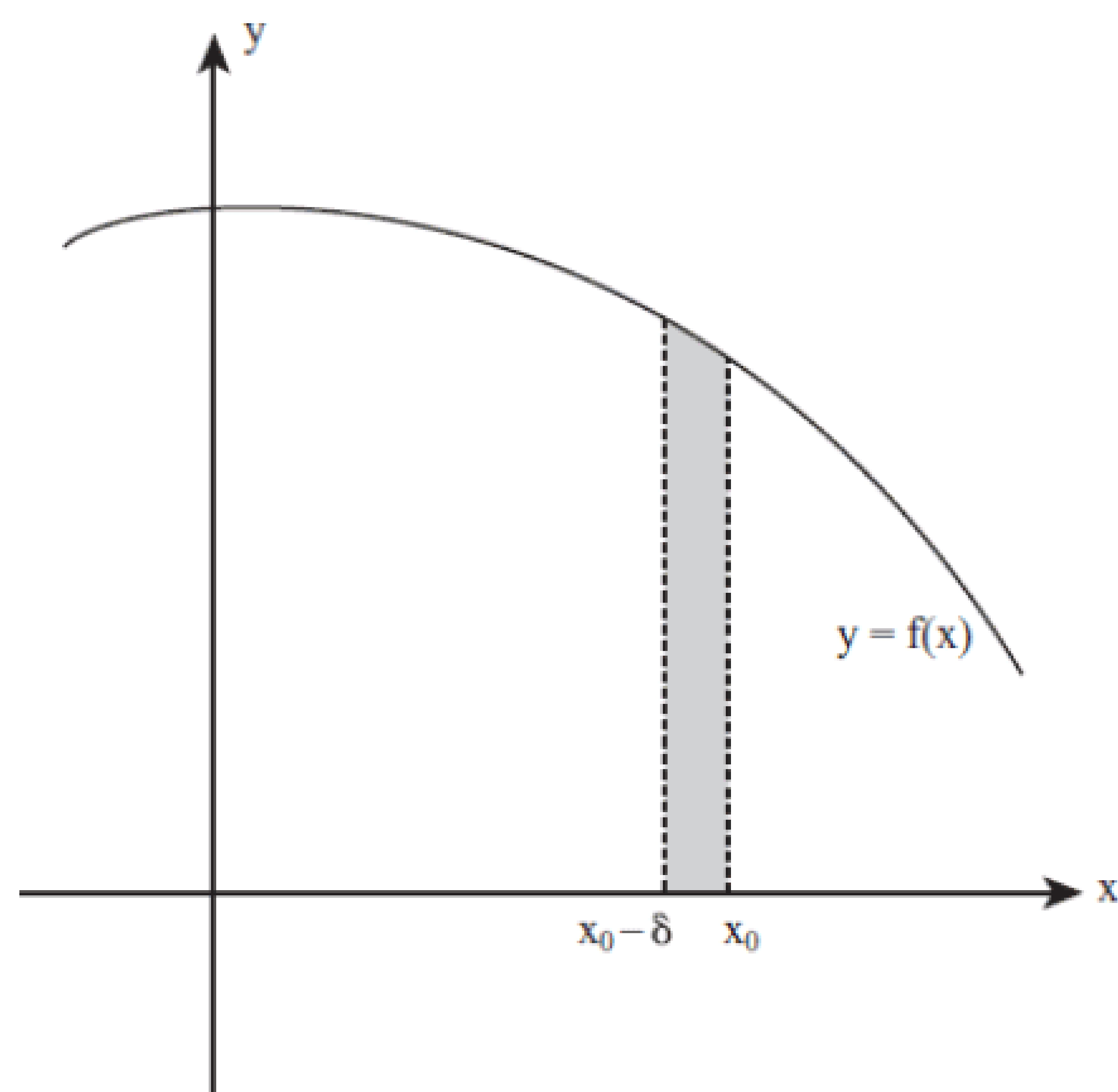


图 6.4　积分计算曲线下方的面积

尽管不定积分表示为一个函数，但也可据此计算特定的面积。在图 6.4 中，灰色区域表示曲线下方两个特定 x 值之间的面积，该面积值称作定积分。定积分表示为一个数值，即面积测定值，而非一个函数。对此，可先期计算不定积分 $f(x)$，并于随后在两个端点处插入 x 值。根据此方案，x_1 和 x_2 之间的定积分等于 $f(x_2)-f(x_1)$。需要说明的是，不定积分所引入的常量值 c 可在当前计算中被消去，最终，定积分处于确定状态。

积分运算采用“∫”符号表示，并通过字母 d 标记积分变量（表示较小的变化量）。因此，函数 $g(x) = 2x-5$ 的积分形式可表示为 $\int g(x)dx = \int(2x-5)dx = x^2-5x+c$，这也意味着，针对 x 的较小变化，函数 g 通过 x^2-5x 方式增长。

定积分采用相同方式加以书写，但需要将主变量的起始值和结束值置于积分符号的两端，进而表明积分区间。下列算式显示了位于[1,3]区间内的定积分：

$$\begin{aligned}\int_1^3 g(x)dx &= \int_1^3 (2x-5)dx \\ &= [x^2-5x]_1^3 \\ &= (3^2-5\times 3)-(1^2-5\times 1) \\ &= -2\end{aligned}$$

上述计算将得到 $y=2x-5$ 下方、$x=1$ 和 $x=3$ 之间的面积。

6.3 微 分 方 程

下列情形常出现于物理计算中，即基于函数 $y(x)$的准确公式尚处于未知状态，而 x，y 以及 y 的导数之间的关系为已知内容，该关系称作微分方程。严格地讲，为了与包含多个变量的偏微分方程加以区别，此类方程称作常微分方程或 ODE。

6.3.1 常微分方程的特征

下列内容显示了常见的常微分方程示例：

- $y'=2x$。
- $(y'')^2+2y=0$。
- $y'-2xy+y^2-x=0$。

如何区分微分方程与其他类型方程之间的不同之处？总体而言，读者尝试求解的问题可视为一类代数解，即 $y(x)$。若不存在对应解，则读者至少可尝试计算（y 的）一个或多个特定值，或与微分方程相近的某一函数。若该函数近似于微分方程，则称作数值解。

微分方程大多难于求解。下面考察形如 $y'=f(x)$ 的微分方程，大多数积分习题均会涉及此类方程的求解过程。对此，除雪机问题可视为一类常见示例。假设除雪机的速度反比于雪层的厚度，当轻触地表上的雪层时，需要计算铲雪机的行进距离。

假设雪层的初始厚度为 s，且降落速率为 σ。由于铲雪车的速度反比于降雪量，因而有如下算式：

$$\frac{dx}{dt}=\frac{k}{s+\sigma t}$$

对等式右侧执行积分运算可得到：

$$x=\frac{k}{\sigma}\ln(s+\sigma t)+c$$

假设 $t=0$ 处有 $x=0$，则有 $c=-\frac{k}{\sigma}\ln(s)$，当前计算演变为：

$$x=\frac{k}{\sigma}(\ln(s+\sigma t)-\ln(s))=\frac{k}{\sigma}\ln\left(1+\frac{\sigma}{s}t\right)$$

根据 x 计算 t，逆转上式后可得到如下算式：

$$t=\frac{s}{\sigma}\left(\exp\left(\frac{\sigma x}{k}\right)-1\right)$$

需要注意的是，上述计算涉及积分未知常数项 c。在问题求解之前，需要了解与此相关的信息，即铲雪车的起始位置。如前所述，这意味着微分方程包含一组解，特定的常量 c 将生成该方程的有效解，这也是微分方程的通用特征。

由于微分方程与向量十分类似，因而可通过系统初始条件方式设置积分参数。该操作并未告知当前具体位置，仅表明“若当前位置于此，则执行有关操作，并于稍后位于另一个位置，进而继续执行对应操作”。

图 6.5 显示了微分方程与向量之间的相似行为。这里，假设微分方程为 $y'-2xy+y^2-x=0$，则可选取任意特定点(x, y)，计算该点处的 y' 值，进而实现微分方程的图形化效果。随后，可在该点处通过适当梯度值绘制一条短直线，重复该步骤多次即可得到如图 6.5 所示的曲线。

图 6.5 中的各条直线表示为 ODE 的特定解，而通解依然包含由积分引入的多个未知参数。需要注意的是，该图体现了一类固定模式。若不考虑选取的初始条件，全部函数 $y(x)$收敛于直线 $y = 2x$ 上的较大 x 值处，进而体现了所选的数据值范围。针对较大的 x 和 y 值，x 中的线性数据项逐渐势微。其他微分方程还将涉及环、奇异点、奇异吸引子等有趣现象。

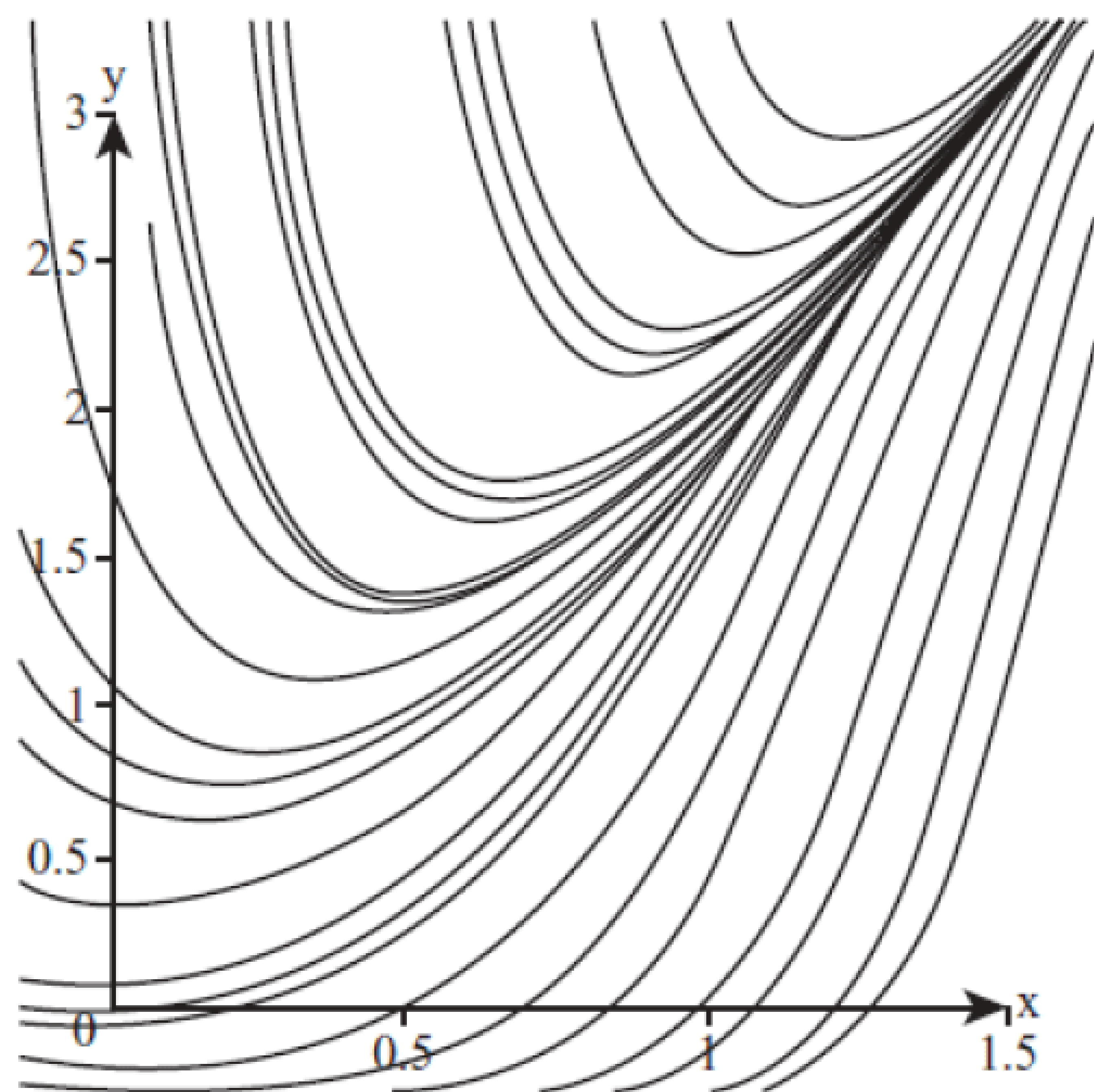


图 6.5　微分方程 $y'-2xy+y^2-x=0$ 的数值标绘图

6.3.2　求解线性 ODE

如前所述，大多数 ODE 无法通过代数方式求解，而有效数值方案却可担此重任，例如线性

ODE，其形式如下所示：

$$f_0(x)y + f_1(x)y' + f_2(x)y'' + \cdots + f_n(x)y^{(n)} = 0$$

其中，$y^{(n)}$表示 y 的 n 阶导数，f 表示为 x 的函数。

当首次考察线性 ODE 示例时，可假设全部函数均为常量，求解方程的关键之处在于 exp() 函数。由于该函数即为自身的导数，因而形如的 $y = \mathrm{e}^{rx}$ 函数其导数为自身的 r 倍。例如，假设微分方程为 $2y - 5y'+3y'' = 0$，读者可尝试解 $y = \mathrm{e}^{rx}$，并查看计算结果。若将该函数返至当前微分方程，则可得到如下算式：

$$2\mathrm{e}^{rx} - 5r\mathrm{e}^{rx} + 3r^2\mathrm{e}^{rx} = 0$$

由于 e^{rx} 为正值，因而析出因子后可得到 $2-5r+3r^2=0$。作为二次方程，对应解为 $r=2$ 或 $\frac{1}{3}$。

另外一方面，由于存在两个工作变量，因而 ODE 的求解过程较为复杂。若有效函数乘以一个常量，即 $y = A\mathrm{e}^{rx}$，则约去因子 A 后，任意倍数 A 均可生成微分方程的有效解。类似地，若指数项中加入常量 c，即 $y = A\mathrm{e}^{rx+c}$，则在微分过程中该常量将会消失，因而不会对有效解产生任何影响。因此，ODE 族包含两个参数并可被初始条件所影响（第 16 章将对此予以深入讨论）。

6.4 近 似 方 案

在积分计算中，近似解可视为一类重要的技术，除了微分方程之外，其他诸多方程也无法采用代数方案进行求解。某些时候，对应方程并不复杂，例如，$\sin(x) = x$ 即涉及一类非代数解。

由于单变量方程可“减至” $f(x)=0$，因而当前问题演变为计算 $f(x)$ 函数的根值——对此，存在大量的可行方案，当处理相对连续的函数 f 时，此类方案已然足够。

6.4.1 划界法

如图 6.6 所示，某些场合下，一类简单方案会涉及定位法或划界法，此类方法源自二分方案。在图 6.6 中，两个 x 值分别位于 x 轴的两侧。这里，存在一种简单方式对此进行检测，即计算 $f(x_1)f(x_2)$。若结果为负值，则二者分别为正、负值；若结果为正值，则二者皆为正值或皆为负值；若结果为 0，则其中一值为最终的根。综上所述，针对某一连续函数，若存在此二值，则根值位于二者之间的某处。

若任意接近某一根值，上述方案可生成一类简单算法。若 $f(x_1)$ 和 $f(x_2)$ 位于 x 轴的两侧，则可考察 $x_3 = \frac{1}{2}(x_1 + x_2)$。通过检测 $f(x_1)f(x_3)$ 和 $f(x_3)f(x_2)$，可将根值位置局限于$[x_1,x_2]$中的某一半区间中。由于均分方法表示为指数过程，因而二分方案可快速地定位根值。除此之外，读者还可从开始阶段准确地确定近似结果的接近程度。

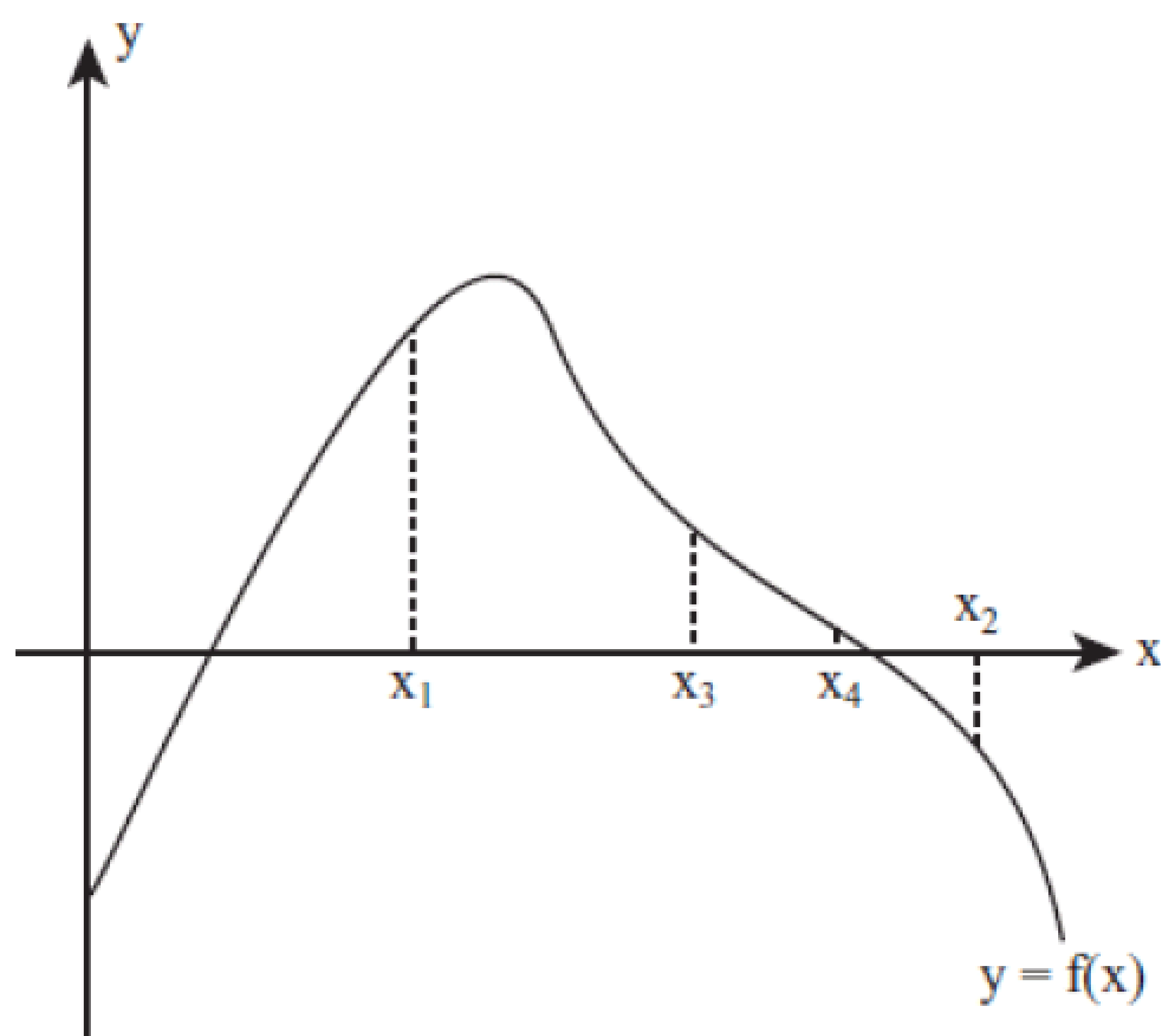


图 6.6　根值定位

bisectionMethod()函数显示了二分法的程序内容，取决于 $f(x)$ 的定义方式，此处并未详细讨论 $f(x)$ 值的计算方式。bisectionMethod()函数的具体内容如下所示：

```
function bisectionMethod(func, x1, x2, resolution)
  //check that f(x1)*f(x2)<0 to get the process going
  set f1 to calculateValue(func, x1)
  set f2 to calculateValue (func, x2)
  if f1*f2>0 then return "may be no root in range"
  //if you are already very near to the solution then return one value
  if abs(x1-x2)<=resolution then return x1
  //(in some circumstances you might choose to return
  //the value of x giving a positive value of f,
  //so you'd return x1 if f1>0, x2 otherwise.)
  set x3 to (x1+x2)/2
  set f3 to calculateValue (func, x3)
  if f3=0 then return x3
  if f1*f3<0 then return bisectionMethod (func, x1, x3, resolution)
  return bisectionMethod (func, x3, x2, resolution)
end function
```

Regula Falsa（无效位置）方案可视为一类快速的划界方法，该方法源自以下事实：通常情况下，可采用最小绝对值计算界值附近处的根值。对此，Regula Falsa 方案通过如图 6.7 所示的方法对新测试点的选取执行加权操作。

假设当前界值为 a 和 b，则可通过 $\dfrac{af(b)-bf(a)}{f(b)-f(a)}$ 计算新值 c，其程序实现方案可视为 bisectionMethod()函数一个简单的变体。

划界法具有其自身的局限性，其中较为明显的是，若函数的两个根值彼此接近，除非初始选择值 x_1 和 x_2 选取得当，否则，通常情况下不会得到两个根值。

若函数存在不连续现象，则划界法同样会失效，例如 $y=x–1$ 函数。该函数在 $x=0$ 处处于不连续状态，并返回非连续点。

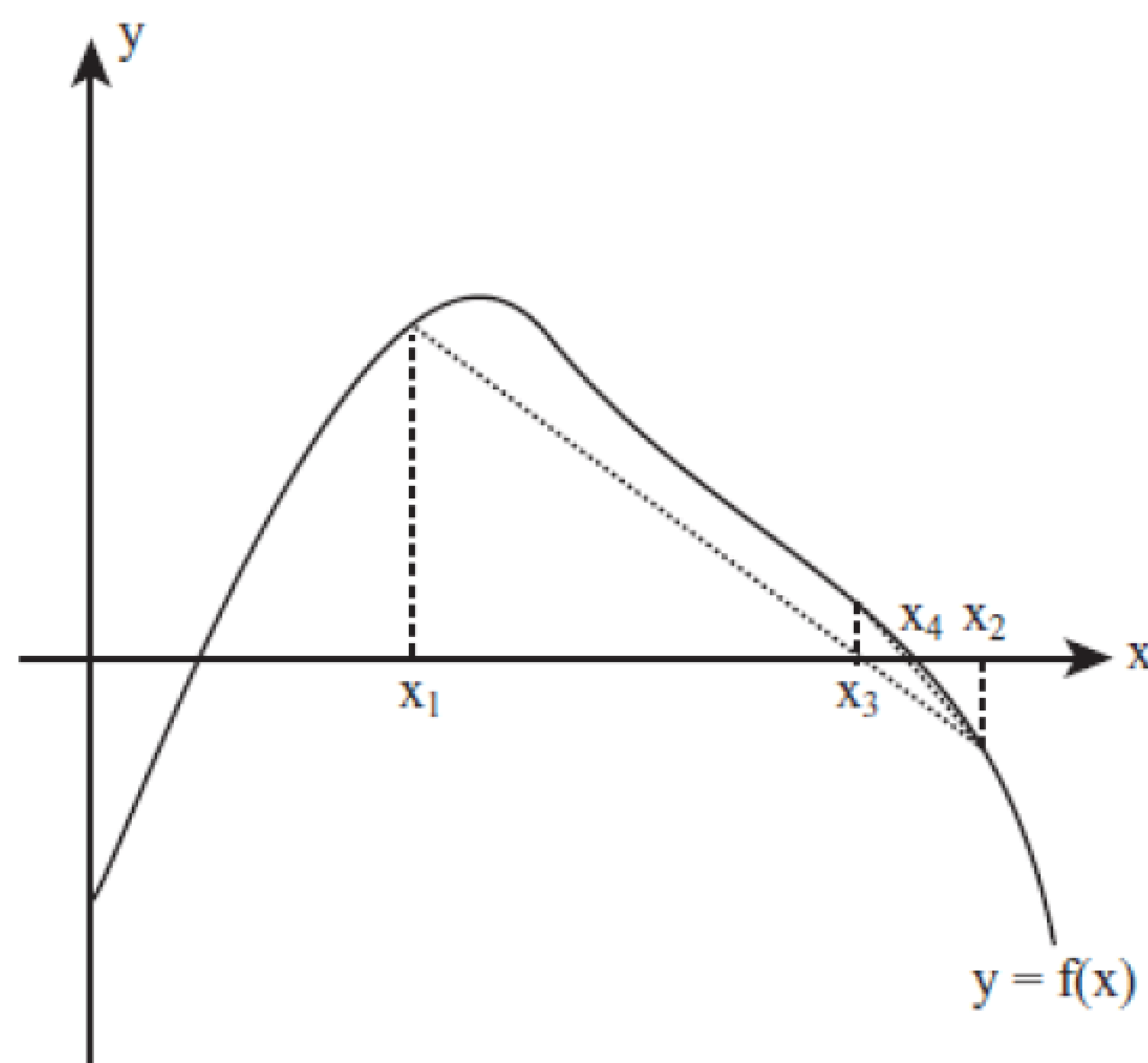


图 6.7　Regula Falsa 方案

针对任意函数，计算合适的初始值通常较为棘手，且需要反复测试。多数时候，若针对对应函数执行初步分析，则划界法尚工作良好。即使 $f(x)$ 函数处于未知状态，该方法依然有效，例如该函数在 x 值处生成任意高度值列表。

6.4.2　梯度方案

针对连续函数（特别是 x 轴附近不包含最大值和最小值的函数），存在一类算法可避免划界法所面临的难点，即 Newton-Raphson 法。该方法采用了如图 6.8 所示的操作技巧。具体而言，该方法使用了特定点处的函数切线，并将其投影至 x 轴上。与原始 x 值相比，切线与 x 轴之间的交点更接近于根值。

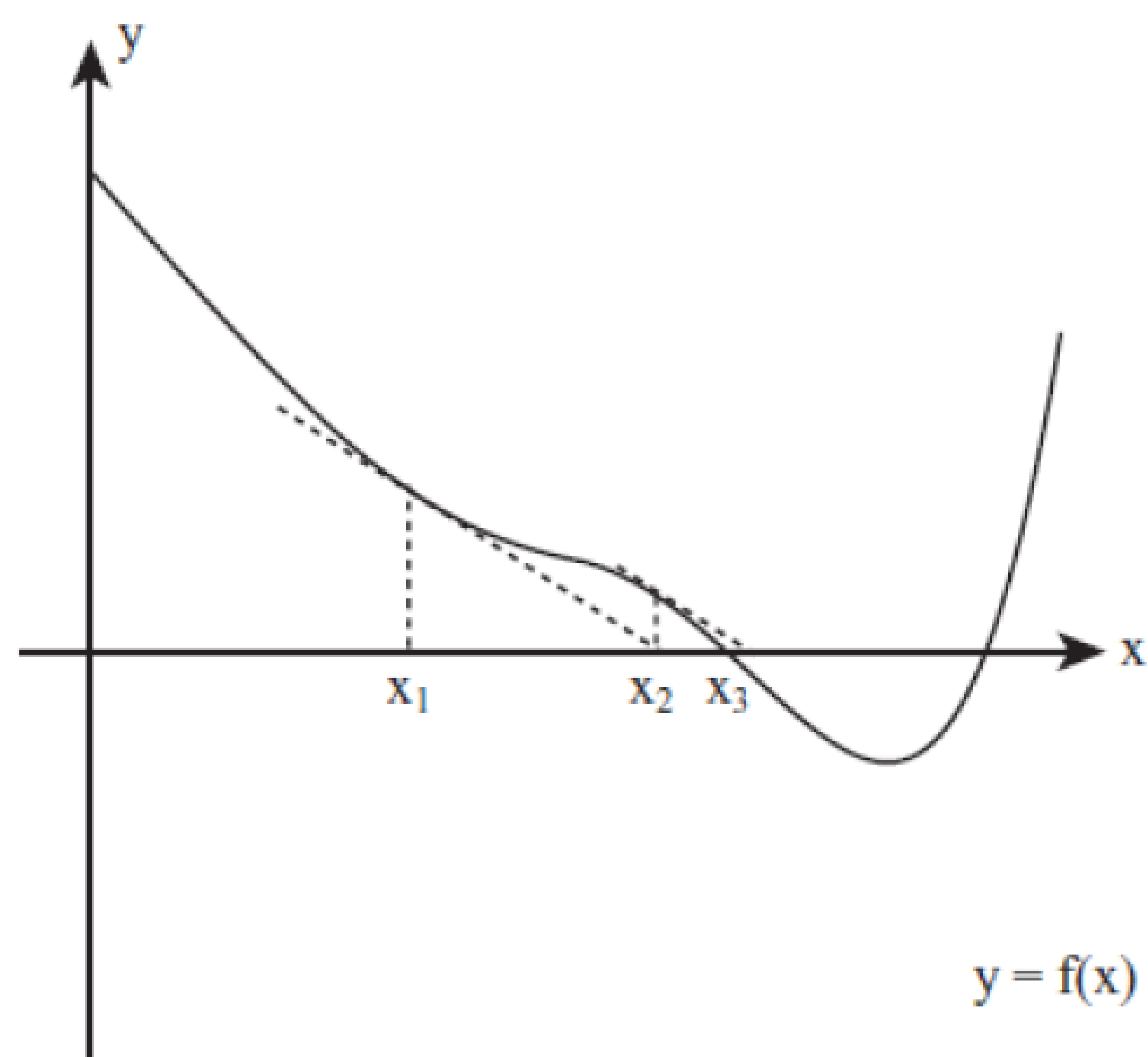


图 6.8　使用函数切线以接近根值

总体而言，图 6.8 并未对相关技巧予以清晰表达，另外，图中所选取的函数也较为特殊。在某种程度上，图 6.9 体现了相关问题：若初始选取的 x 值位于折点附近，或者 x 和根值之间存在折点，则问题也会随之产生。然而，若 x 的初始值足够接近某一根值（对于大多数函数而言，该距离依然较大），则该方案工作良好。而且，与其他常见方法相比，Newton-Raphson 法具有较快的计算速度。

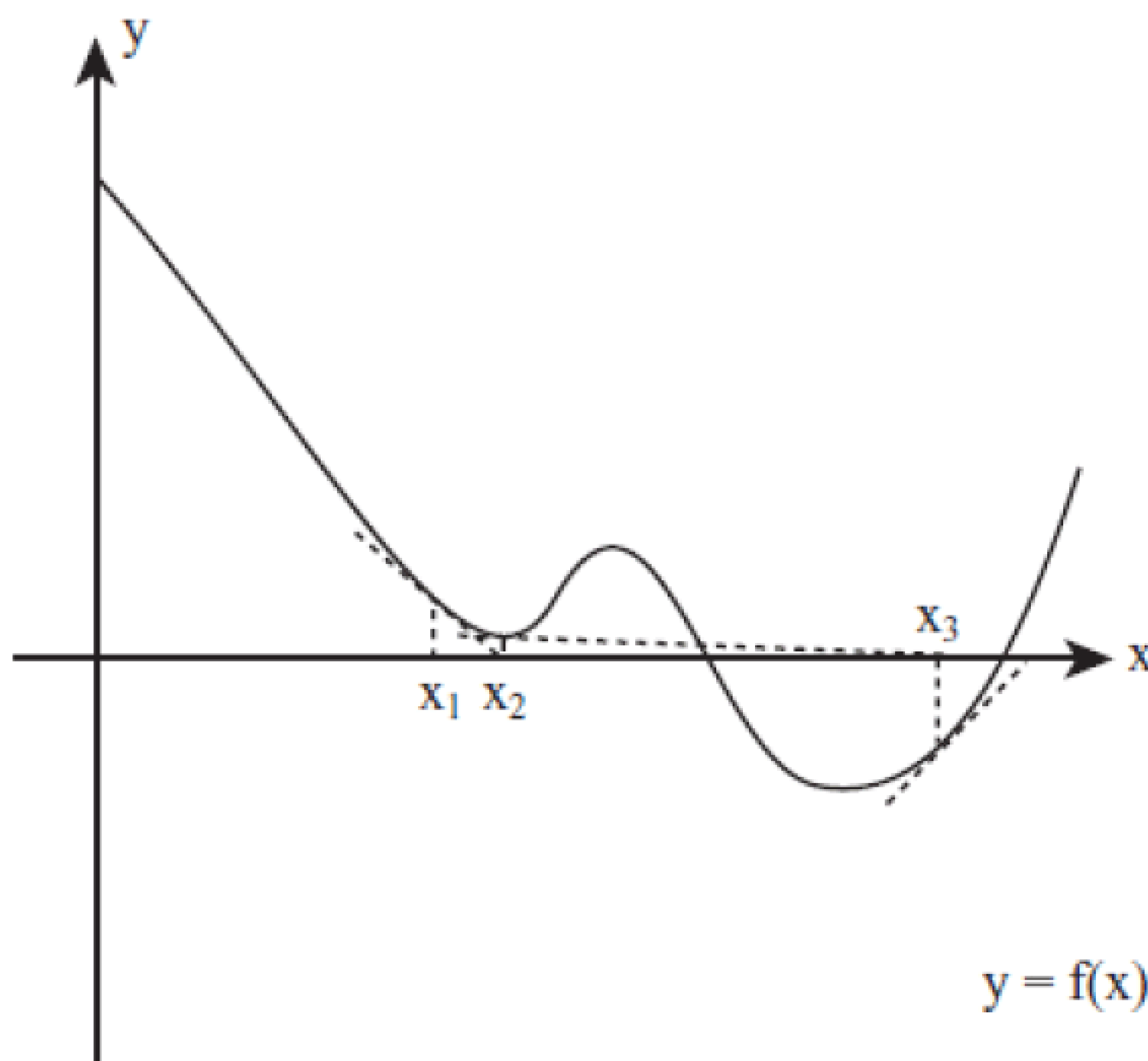


图 6.9 Newton-Raphson 无法有效地计算附近根值

Newton-Raphson 法相对简单，并可通过标准的函数图像技术予以计算。位于 x_1 处的切线梯度表示为 $f'(x_1)$，因而直线方程为 $y-f(x_1)=f'(x_1)(x-x_1)$。该法方程将在 x_2 点穿越 x 轴，其中，$x_2=x_1-\dfrac{f(x_1)}{f'(x_1)}$。

newtonRaphson()函数针对 Newton-Raphson 法提供一类迭代方案，且易于通过编程方式加以实现，如下所示：

```
function newtonRaphson(func, deriv, x1, resolution)
  set f1 to calculateValue (func, x1)
  if abs(f1)<resolution then return x1
  set g1 to calculateValue (deriv, x1)
  if g1=0 then return newtonRaphson(func,
     deriv, x1+resolution, resolution)
  set x2 to x1-f1/g1
  return newtonRaphson(func, deriv, x2, resolution)
end function
```

与其他示例不同，若函数导数未知，则 Newton-Raphson 法通常难以实施，这也可视为一种缺陷，针对某些复杂函数或随机函数尤其如此。

基于梯度的最后一个示例是割线法，该方案具有较少的编码量，因而优于 Newton-Raphson 法，如图 6.10 所示。

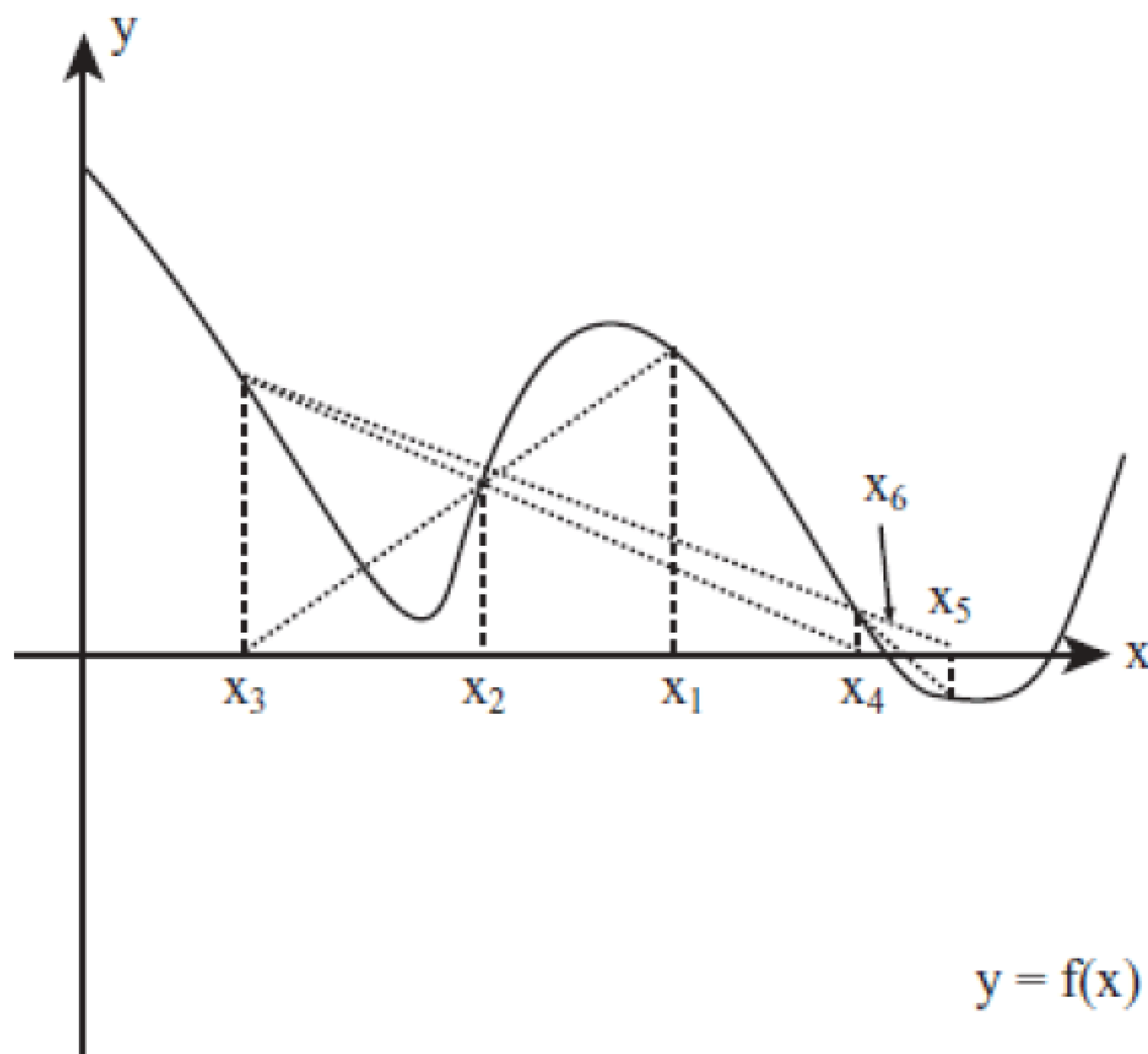


图 6.10　割线法

从本质上讲，上述处理过程等同于 Regula Falsa 法，仅是移除了异号 $f(x)$ 值这一情形。相反，该过程假设函数梯度在测试点附近呈线性状态，类似于 Newton-Raphson 法，读者可尝试将直线外推至 x 轴处。

再次强调，对于在 x 轴附近包含折点的函数，割线法同样会产生问题。类似于 Newton-Raphson 法，该方案会遗失起始点附近的根值。无论如何，若全部工作仅为计算某一根值，则割线法具有稳定、快速等特征；相比较而言，若需要在某一特定区域内获取根值（例如[0,1]区间内），则建议使用划界法。

针对近似方案的程序设计，应确保包含某些检测操作和平衡措施，这将涉及迭代计算，因而应避免无效根值所引起的无限循环。

6.5　本 章 练 习

【练习 6.1】 试编写函数并生成与图 6.5 类似的微分方程函数图，该函数的主体思想源自第 3 章中的 drawGraph()函数。

【练习 6.2】 试编写函数以实现割线法近似方案，该函数可视为本章 bisectionMethod()函数的直接扩展。

6.6　本 章 小 结

本章引入了与微积分运算相关的关键技术。尽管积分或微分方程的求解并非是本书的重点内

容，但读者依然有必要理解相关概念，并为后续章节的学习打下坚实的基础。因此，本章重点讨论微积分背后的诸多理论。同时，本章还显示了基于微分方程的应用程序，进而求解某些物理问题。本书第 2 部分尝试将理论与实践相结合，并以此模拟物理学中的真实运动行为。

至此，读者应掌握如下内容：

- 术语“微积分”、“导数”、“偏导数”以及“不定积分”的含义。
- 简单函数的微积分计算，例如多项式。
- 指数和三角函数的导数计算。
- 微分方程的识别方式，以及简单示例的求解方案。
- 针对方程的近似解，采用划界法和梯度方案进行计算。除此之外，读者还应了解不同方案之间的优、缺点。

第2部分　物理学基本内容

本书第2部分将讨论基本的物理运动，包括真实世界中的粒子运动方式及其程序设计。另外，此处核心内容仍为向量，据此，读者可将复杂问题划分为多个简单问题。最后，相关章节还将分析撞球游戏的构造过程。

第 7 章　加速度、质量和能量

本章包含如下内容：

- 概述。
- 弹道学。
- 质量和动量。
- 能量。

7.1　概　　述

本章将暂别抽象的数学内容，转而进入物理对象世界，即力学。尽管如此，本章内容依然难以完全脱离前述章节所讨论的知识。例如，第 5 章曾考察了向量运动的简单示例，并引入了速度、速率、位移、距离以及时间等概念，本章还将进一步分析加速度这一概念。当采用加速度时，则可模拟重力作用下的粒子运动行为。

7.2　弹　道　学

在工程物理领域中，弹道学主要研究恒定加速度状态下的粒子运动，多数时候，粒子可视为一类发射对象，此类对象穿越空气并在重力作用下坠落。严格地讲，空气中运动的物体还会受到阻力的作用，其深入讨论则超出了本书的范围。当发射物体穿越空气介质时，任何影响该物体速度的事物均会对其加速度产生影响。

7.2.1　加速和减速

速度表示为粒子位置的变化率，加速度则表示速度的变化率。类似于速度，加速度也表示为一个向量，各分量采用速度单位除以时间单位进行描述。例如，距离单位为米，时间单位则为秒。另外，距离和位移均采用米加以测定，速率和速度通过米/秒定义，加速度则采用米/秒 ^{2} 予以确定。

【提示】与速率/速度以及距离/位移不同，加速度不存在对应的标量描述，并可同时应用于速率变化率（标量）和速度变化（向量）。总体而言，其应用与具体的上下文环境相关。

由于加速度表示为向量，因而涉及方向计算。若粒子直线移动，且速度以恒定速率减少，则该粒子受到反向恒定加速度的影响。这里，运动行为似乎与“加速度”一词不符，后者常使人联想到速度的增加。例如，如汽车的行驶速度增加，则车辆处于“加速”状态。相应地，读者可将减速视为逆加速度。在物理学中，“加速度”一词则更为常见。

7.2.2　基于恒定加速度的运动方程

若采用 **a** 和 **u** 表示向量，且粒子的加速度 **a** 定义为常量，则 **a** 将引发对象的速度变化。若粒子的初始速度表示为 **u**，则历经时间 t 后，速度 **v** 等于 $\mathbf{u}+\mathbf{a}t$。

同时，对象的位置计算也较为简单，并可采用时间段 t 内的平均速度。若时刻 t 处的速度为 **v**，则对应的平均速度表示为$(\mathbf{u}+\mathbf{v})/2$，位移 **s** 为 $t(\mathbf{u}+\mathbf{v})/2$。替换前述 **v** 值后则可得到如下算式：

$$\begin{aligned}\mathbf{s} &= \frac{t(\mathbf{u}+\mathbf{v})}{2} \\ &= \frac{t(\mathbf{u}+\mathbf{u}+\mathbf{a}t)}{2} \\ &= \mathbf{u}t+\frac{1}{2}\mathbf{a}t^2\end{aligned}$$

第 6 章曾讨论到，若根据时间执行微分运算，则可得到如下算式：

$$\frac{d\mathbf{s}}{dt}=\mathbf{u}+\mathbf{a}t=\mathbf{v}$$

$$\frac{d^2\mathbf{s}}{dt^2}=\mathbf{a}$$

这与速度和加速度定义相吻合。速度表示为位移的变化率，加速度则表示速度的变化率。需要注意的是，若加速度为 0，则公式演变为基于恒定速度的简化版本，即 $\mathbf{s}=\mathbf{u}t$。

当前 3 个方程已知，并与 **s**，**u**，**v**，**a** 和 t 关联，各方程将 5 个变量中的 4 个量值进行关联。通过适当的替换操作，读者还可得到更多的方程。例如，第 4 个方程将 **v**，**u**，**a** 和 **s** 关联，第 5 个方程将 **v**，**a**，**s** 和 t 关联。最终，全部 5 个变量方程如下所示：

（1）$\mathbf{v}=\mathbf{u}+\mathbf{a}t$

（2）$\mathbf{s}=t(\mathbf{u}+\mathbf{v})/2$

（3）$\mathbf{s}=\mathbf{u}t+\frac{1}{2}\mathbf{a}t^2$

（4）$\mathbf{s}=\mathbf{v}t+\frac{1}{2}\mathbf{a}t^2$

（5）$\mathbf{v}^2=\mathbf{u}^2+2\mathbf{as}$

由于向量乘法稍显特殊，因而基于向量的公式（4）缺乏应有的实际意义，该方程仅体现了标量版本，并假设 **s**，**a**，**v**，**u** 处于非共线状态。另外，该方程仅是前述方程的重构版本，不难发现，方程（3）和（4）可通过公式（1）和（2）推导得出。

当采用上述公式时，可根据其他 3 个变量计算某一参数。此处，假设速度和加速度为已知项，并计算粒子的移动距离。对此，可计算粒子的相对初始位置和初始速度，如下所示：

若 $\mathbf{v} = \mathbf{u} + \mathbf{a}t$

则 $\mathbf{u} = \mathbf{v} - \mathbf{a}t$

且 $\mathbf{s} = t(\mathbf{u} + \mathbf{v})/2$

该计算通常较少出现。由于 **u**，**a** 和 *t* 通常为已知内容，因而往往需要计算速度和位置数据，这也是上述各项公式的用武之地。

作为获取位置数据的一类通用解决方案，calculatePositon()函数接收 4 个参数，并返回位置数据，如下所示：

```
function calculatePosition(initialPosition,initialVelocity,acceleration, time)
  return initialPosition + initialVelocity * time + acceleration * time * time/2
end function
```

7.2.3 基于重力的加速度

伽利略为了了解重球和轻球是否以不同速度下落，则登上了比萨斜塔并将两个球体同时掷下。最终，二者以相同的速度下落。通过该试验，伽利略证明了对象的下落速度与其重量无关。实际上，伽利略从未尝试过此类试验，相反，他采用了纯逻辑方式推导出了上述定理，其过程颇具独创性，下面不妨对此予以考察。

假设存在两个对象 A 和 B，且 A 的重量大于 B。当前，假设 A 的降落速度大于 B 并以此为真。若将 A 与 B 连接在一起，并从某一高度下落，则 B 的下降速度较慢，因而对 A 形成阻力，这一点与降落伞有几分类似。这也意味着，若 A 和 B 连接在一起，则降落速度小于 A。然而，该结论并不正确：A 和 B 连接后的重量大于 A，则下降速度也应大于 A，因而原假设错误。

【提示】在数学领域内，上述证明称作反证法，即相反结论导致矛盾的出现，进而证明原结论正确。与直接证明相比，尽管某些人士认为反证法缺乏一定的优雅性，但不可否认的是，该方法功能强大。

球体在空气中的降落行为究竟如何？重力饰演了何种角色？实际上，此类问题颇为复杂，当前，读者仅需了解一种相对简单的答案。若球体的海拔高度未发生显著变化，则该对象将受到一个恒定的向下加速度的作用。其中，加速度源自地球引力，若忽略空气阻力，则可视为球体速度和方向的唯一变化方式。在地球海平面处，$|\mathbf{g}|$约为 9.8m/s^2，为了简化计算，在本章中，该值定义为 10。

这一举措使得问题的求解方式更加直观，例如，若径直向上抛出一个球体，且初始速度为 5ms^{-1}，试计算球体最终到达的高度。当球体到达最高点时，其速度为 0，对应运动方程如下所示：

$$v^2 = u^2 + 2as$$

$$0 = 52 - 2\times10\times s$$

$$s = 25/20 = 1.25\text{m}$$

这里，*s*，*u*，*v*，*a* 均为向量值，在当前计算中，读者可将其视为普通的数字。此处唯一保留的向量特征是 **g** 采用了负值，即−10，其原因在于，**g** 与其他量值呈反向关系。具体而言，全部运动发生于一条独立的垂直线上，且位于一维空间内，当前向量仅包含一个分量，即一个有向

数值。

又如，当球体落回手中时，其速度又当如何？对此，可计算 v 且满足 s=0，对应方程如下所示：

$$v^2 = u^2 + 2as = 25 + 0$$

求解后，$v = \pm 5$。

$v = 5$ 表示为运动的初始阶段，且根据定义有 $s = 0$。在运动结束时，$v = -5$，即球体落回原处但运动方向相反。因此，只有理解了数据的方向特征，方可正确地使用运动方程。

为了进一步理解上述方程的应用方式，下面考察最后一个示例。若球体的抛出速度为 5m/s，则经过多长时间后，其位移为−1.5m？对应方程如下所示：

$$s=ut+\frac{1}{2}at^2$$

$$-1.5=5t-t^2$$

$$t^2-t-0.3=0$$

$$t=\frac{1\pm\sqrt{1+1.2}}{2}$$

最终，t 为 1.2s 或−0.2s，此处可忽略负值（忽略负值的意义在于，若根据时间值反向考察当前球体投射行为，则球体被地表所投射，而非手掌），即球体抛出后经历了 1.2s。

7.2.4　炮弹的运动行为

若球体并未径直向上抛出，情况又当如何？此时，球体的速度和加速度不再共线，因而无法实现单值计算。对此，可采用第 5 章所讨论的技巧，进而将速度分为多个分量。若坐标系分别采用水平向量$(1\ 0)^T$和垂直向量$(0\ 1)^T$，则基于重力的加速度仅作用于某一基向量上。这也意味着，水平方向上的运动分量不会受到重力的影响，即水平速度保持恒定，仅垂直方向上的运动分量存在加速度。

针对炮弹对象，火炮可视为一类炮弹发射装置，并包含一定的发射角度和发射速度。其中，炮膛为一类管状对象，并推动炮弹沿特定路径运动。炮弹的发射速度可根据火药量以及炮弹重量进行适当的校准（这也是能量守恒的一个例子，稍后将对此加以讨论）。

如图 7.1 所示，若火炮与地面间的角度为 θ，且炮弹的合成速度表示为 u，则炮弹初始速度的水平分量为 $u\cos(\theta)$，垂直分量为 $u\sin(\theta)$。而且，若炮管的长度为 l，则炮弹出膛后的高度值为 $l\sin(\theta)$，读者可据此计算炮弹的飞行距离。此处，假设地表呈平坦状态，因而炮弹垂直方向上的行进距离为$-l\sin(\theta)$，并可忽略火炮的高度及其运动的水平分量。通过前述内容可计算炮弹在空中的飞行距离，对应方程如下所示：

$$s = ut+\frac{1}{2}at^2$$

$$-l\sin(\theta)=u\sin(\theta)t-5t^2$$

$$5t^2-u\sin(\theta)t-l\sin(\theta)=0$$

$$t=\frac{u\sin(\theta)+\sqrt{u^2+\sin^2(\theta)+20l\sin(\theta)}}{10}$$

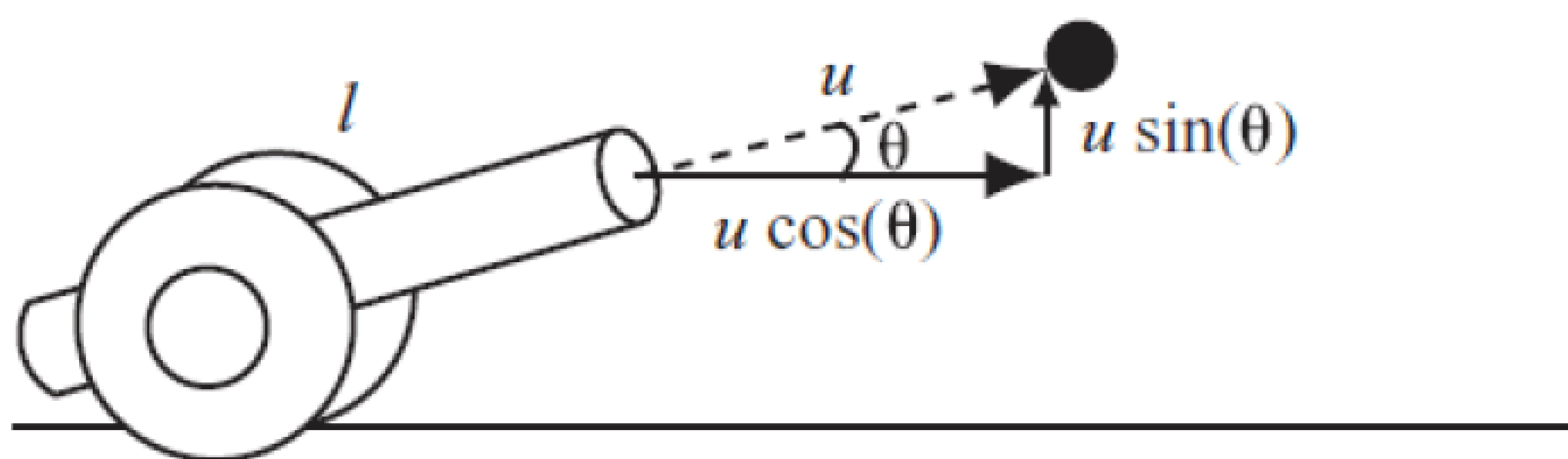


图 7.1　火炮以特定的角度发射炮弹

上述方程稍显复杂，为了简化当前问题，此处可对某些数值进行适当替换。这里，假设 θ=30°，因而 sin(θ)=0.5。类似地，若炮管长度为 2m，且炮弹以 20m/s 的速度发射，则飞行时间如下所示：

$$t=\frac{20\times0.5+\sqrt{20^2+0.5^2+20\times2\times0.5}}{10}$$

$$=1+\frac{\sqrt{120}}{10}2.1\text{s}$$

读者可使用上述信息以及（恒定）水平速度计算飞行距离，如下所示：

$$s=ut=u\cos(\theta)t=20\times\frac{\sqrt{3}}{2}\times2.1=36.3\text{m}$$

当采用微积分计算时，可能需要计算炮弹飞行最大距离所需的角度值（读者可运用当前所学知识尝试对该问题进行计算）。出于简单考量，可忽略火炮的高度，并假设与地面碰撞时其垂直位移为 0。在练习 7.2 中，读者可尝试编写相关函数，并对火炮进行调校，进而使发射对象击中特定点。

此类技术较为通用，并可计算平台游戏中人物角色的跳跃行为，以及射击范围内子弹的飞行状态。其中，发射物的初始速度较为关键，7.3 节将对此予以讨论。

7.3　质量和动量

截止到目前为止，粒子重量尚未予以考虑。也就是说，当发射对象处于行进状态时，其飞行路径与其重量无关。然而，根据经验可知，不同重量的对象，其飞行状态截然不同。本节将对质量加以考察，并对砖块和乒乓球这两种难易程度不同的投掷行为加以解释。

7.3.1　质量和惯性

重量包含两种描述方式，包括日常会话中的重量，以及物理学范畴内的作用力，该作用力源自重力并可视为作用于物体上的、方向向下的作用力，其（向量）量值可通过计重秤加以测定。另外一个术语是质量或惯性，对此，需要依据作用力“语言”对其进行准确描述。在当前上下文

环境中，读者可将质量作为一种测量手段，进而描述“移动对象 X 时的困难程度”。质量定义为一个标量值，并以克、千克或磅作为计量单位。这一类质量单位常显示于计重秤上，毕竟，若人们生活在同一个星球上，则重量和质量呈正比关系。

粒子的质量通常保持恒定，然而，对于火箭而言，该结论并不正确，具体内容将在第 15 章进行讨论。更为重要的是，与重量不同，无论粒子位于何种环境下（例如中子星或外太空），其质量保持不变。例如，在月球上，对象的重量约为地球质量的 1/6，但其质量未发生改变。因此，若期望达到相同的速度，则应施加相同的作用力。

那么，质量究竟代表了何种含义？一种解释是，质量是物质的基本属性，且向下可追溯至原子级别（或更为深入的微观世界）。而另一种相对简单的解释可描述为，质量体现了对象中所包含的物质，这也意味着，对象的质量在其全部体积内均匀分布。截止到目前为止，相关对象仅涉及抽象的点状对象，例如粒子，若质量遍布于较大空间内的，则对应物体是否具有相同的运动行为？

一类近似方案可描述为（暂不考虑旋转），在对象内部的特定点处，对象的运动路径与点粒子相同，该特定点称作质心或重心。相应地，质心的实际位置与对象的形状和内部质量分布状态有关。对于对称物体，其质心即为对象的中心位置，具体内容将在第 13 章加以讨论。

7.3.2　动量计算

当与其他数据值结合使用时，即可看到质量计算的重要性。例如，动量可视为一类较为重要的数据，并定义为粒子质量和速度的乘积。作为向量和标量的乘积结果，动量也表示为一个向量并与速度平行。

类似地，可将动量视为“停止对象 X 时的难易程度”的一种测定方法。例如，与行驶速度为 15km/h 的玩具车辆相比，令行驶速度为 1km/h、质量为 10t 的卡车停止则困难的多。与此形成鲜明对此的是，这也是为何大力士可用牙齿拉动一辆卡车的原因。尽管令对象运动起来较为困难，然而，一旦该对象处于运动状态，其自身动量则可维持这一状态。

关于动量，守恒问题值得关注。若粒子（或整体粒子系统）未受到外部影响，则全部动量保持不变。

当计算炮弹的运行轨迹时，动量似乎未曾参与其中。毕竟，炮弹对象受到了外部作用力的影响，即重力。也就是说，炮弹在空气中运动时的动量随时发生变化。然而，这一现象仅出现于垂直动量方向上，在与重力垂直的水平方向上，速度和动量均保持不变。当考察碰撞问题时，读者将会进一步了解动量并非是空穴来风。

7.4　能　　量

本小节将介绍能量这一概念，并以不同方式再次考察弹道运动问题。

7.4.1 能量类型

能量体现了对象对其环境的改变程度，具有较大能量的粒子将对其所处环境产生更为强烈的影响。例如，若一发炮弹处于闲置状态，则其影响力有限；而高速飞行的炮弹则可摧毁坚硬的墙壁。能量表示为标量值，并采用焦耳（J）作为计量单位，即 1kg • ms^{-1}（1 千克 • 米/秒）。除了焦耳之外，另一种较为常见的能量单位则是卡路里，但该术语多限于食物摄入问题。

能量包含多种形式，具体内容如下所示：

- 动能。任意处于运动状态下的对象均包含动能，若质量为 m 的粒子其速度为 $\mathbf{v}$，则动能定义为 $\frac{1}{2}m|\mathbf{v}|^2$。不难发现，该式与动量方程关系紧密。
- 重力势能。宇宙万物皆具有此类能量形式，其形成过程较为复杂。针对初学者而言，该术语往往具有一定的相对性。例如，对象甲比对象乙多 5J 势能，但“某一对象具有 5J 势能”这一类说法则较少被提及。对象的重力势能体现了一种潜在的下落趋势，在前述炮弹示例中，若炮弹置于垛口而非地面上，其势能将会对垛口下方士兵产生较大影响。若某一质量为 m 的粒子位于另一粒子上方 h 处，则相对势能表示为 $mh|\mathbf{g}|$。其中，$\mathbf{g}$ 表示为基于重力的本地加速度。为了简化重力势能计算，通常可将某一特定高度定义为 0，并根据该点计算势能。
- 弹性势能。弹性势能存储于可变形胶质对象或弹簧中，并在释放过程中弹射某段距离，第 16 章将对此加以讨论。
- 热能。通俗地讲，热能体现了一种灼伤程度。通常情况下，热能可通过一段时间内发热对象所释放的能量值予以考察。当谈及功率时，热能则通过瓦特（W）进行测算，即焦耳/秒。
- 化学势能。此类能量存储于化学反应物质中，例如炸药，并涉及反应前后稳定性变化所产生的能量。例如，铁和空气反应后可得到氧化铁或铁锈。与纯铁相比，铁锈则表现得更加稳定。其中，锈化过程将释放能量。
- 电能。当电流通过对象时，常会产生电能。电池上常标有电压值，例如 10V，进而表明 1A 电流产生的功率（瓦特）。

尽管能量形式多种多样，但全部能量仅可通过两种方式予以存储。方式一是动能并与运动行为有关。例如，热能可视为物质内原子和分子大规模运动的结果；电能则可视为导体内电子（或其他带点粒子）的运动结果。

方式二则是势能。当环境发生变化时，势能体现了对象的行为能力。势能通常具有一定的相对性，例如，电势能取决于特定的化学反应；重力势能则与所处高度有关；弹性势能则稍显不同，其行为多取决于弹簧的属性。

7.4.2 能量守恒

若未受到外力作用，能量在系统内保持守恒，这一点与动量十分类似。实际上，在某些场合，

即使存在外部作用力，能量依然处于守恒状态，且此类作用力涵盖于能量计算中，重力即是这方面的一个例子。进一步讲，由于通过重力势能方式考察重力，因而重力并非是真正意义上的外力（全部势能均与某种作用力关联）。

能量守恒可计算场景中的不同部分（并采用不同的技术方案）。例如，在前述炮弹示例中，问题之一便是无法求解炮弹的离膛速度。通过能量守恒以及其他已知项，则可从理论上计算该速度。

火药与空气中氧气之间的化学反应形成了炮弹的外部推动力，同时，该反应也将火药的化学势能转变为热能，并生成空气分子的动能以使其快速膨胀。如图 7.2 所示，膨胀效应将动能传递至炮弹。这也意味着，若可计算火药燃烧后所释放的能量，则可进一步计算炮弹所蕴含的能量。

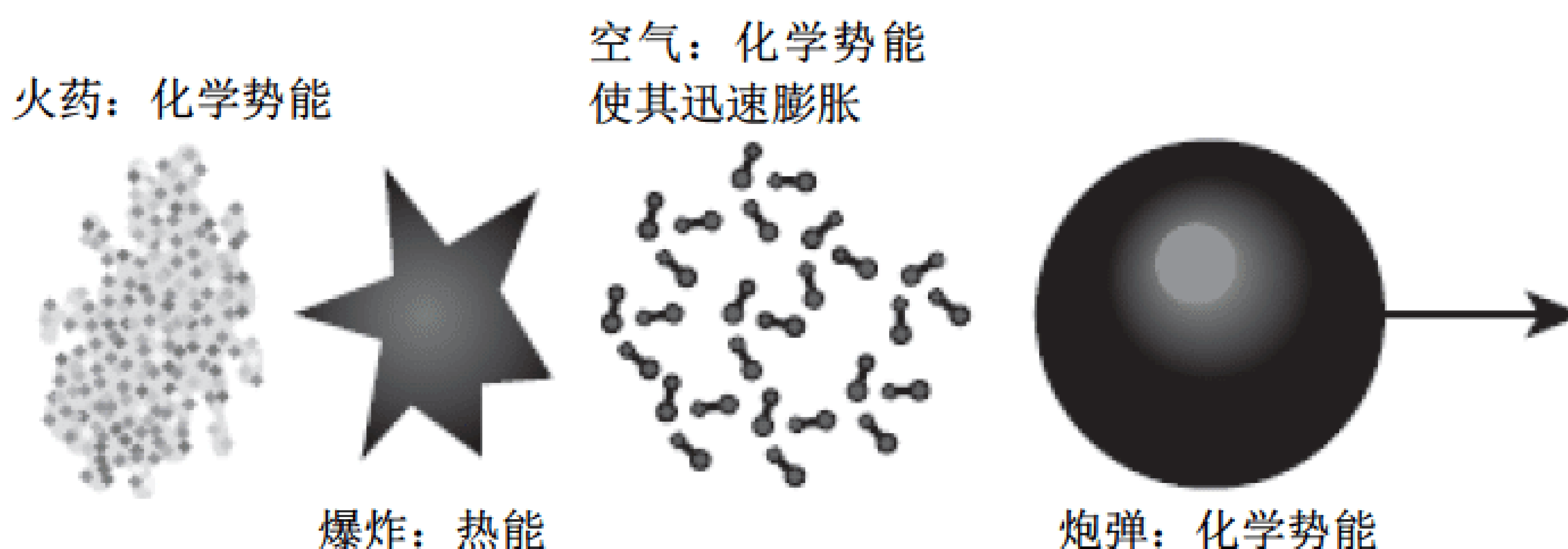


图 7.2　火炮发射时的能量转换

当然，在真实世界中，能量转换并非完美无缺。例如，火药所产生的热量大部分被周围空气所吸收，也就是说，能量转换并非百分之百完成。对此，读者可尝试计算其转换效率。

需要注意的是，炮弹被化学反应所驱动后，根据动量守恒定律，火炮自身向后移动。这里，火炮内部的爆炸效果可视为“内力”。若火炮和炮弹的全部水平动量在发射之前为 0，则发射之后二者之和仍然为 0（该过程并不适用于水平方向，其中，重力作用于炮弹和火炮上）。最终，若炮弹在爆炸后具有前向动量 p，则火炮须有动量 $-p$ 并予以抵消。

这里，假设炮弹的质量是 2kg，火炮的质量是 200kg。待发射后，炮弹的水平速度为 30m/s，其动量为 60kg • m/s。因此，火炮在相反方向具有相同的动量，对应速度为 $\frac{60}{100}$=0.6m/s，即后座力效果。需要注意的是，由于火炮自身重量远大于炮弹，因而其速度小于炮弹的发射速度。

另一个待考察项是爆炸过程中的动能。出于简单考量，此处假设火炮以水平方向发射炮弹，炮弹发射时的动能为 $0.5\times2\times30^2 = 900$J，火炮的动能为 $0.5\times200\times3^2 = 9$J。虽然两个方向上的动量相等，但速度差别使得二者的能量显著不同。其中，火炮的能量远小于炮弹。若火炮于一端固定，则炮弹的发射速度将得到显著提升。也就是说，火炮的后向移动能量使得炮弹产生了附加动能，因而炮弹的动能约为 909J，而非 900J，对应速度约为 $\sqrt{909}$ =30.1m/s。

【提示】 当火炮于一端固定后，动量守恒定律为何不再适用？由于固定端向火炮施加了额外的作用力以使其原地静止，这可视为牛顿第三定律所描述的示例，具体内容可参考第 12 章。

7.4.3　利用能量守恒求解弹道问题

针对弹道问题，能量守恒定律赋予了不同的求解方式。若忽略空气阻力，发射对象在行进过

程中，其能量需保持恒定。具体而言，当该对象处于上升过程中，动能转换为重力势能；当其降落时，势能将转换为动能，读者可据此执行前述计算。例如，可计算炮弹的上升高度。

如前所述，假设火炮以 20m/s 发射炮弹，且炮弹的质量为 m，因而初始动能为 400m。若从炮口处计算势能，则炮弹发射处的势能为 0。因此，在炮弹飞行的整个阶段中，全部能量为 400m（动能和势能之和）。

假设炮弹径直向上发射，在最高点处，炮弹的动能为 0，因而全部势能为 400m。根据势能方程，有 mgh=400m。据此可知，炮弹相距炮口的最高点为 40m。需要注意的是，等式两侧可消去质量一项，因而最大高度值与质量无关。

【提示】此处，变量 m 表示为质量，且不应与长度单位“米”混淆，在本书中，斜体字母表示为变量。

若炮弹以某一角度发射，则可分别在水平方向和垂直方向处理其运动行为。假设炮弹与水平方向呈 30° 角发射，因而其恒定水平速度为 $20\times\frac{\sqrt{3}}{2}=17.3$m/s。在最高处，垂直速度为 0，但水平速度保持不变，因而动能为 $\frac{1}{2}\times m\times1.73^2=150m$J。鉴于全部能量为 400$m$，因而可得到如下算式：

$$150m + mgh = 400m$$

$$gh = 250$$

$$h = 25\text{m}$$

上述各值可适当进行调整，进而考察地面高度差所带来的影响。另外，读者还可通过运动方程求解该问题，并查看是否可得到相同的结果。

尽管能量守恒和动量守恒并未涉及时间项，但在多数时候，该定律可简化计算过程。

7.5 本章练习

在下列练习中，相关变量的计量单位分别为米、秒和千克，且 **g** = 10。

【练习 7.1】 试编写函数 javelin(throwAngle, throwSpeed, time)，该函数计算标枪在一段时间内的位置和角度。

该函数应返回一个二值数组，其中，向量（数组）表示标枪以某一角度投掷后的、时刻为 time 时的位置。在标枪的飞行过程中，其方向为曲线的切线方向。也就是说，对应方向平行于速度向量。

【练习 7.2】 试编写 aimCannon(cannonLength,muzzleSpeed,aimPoint)函数，该函数返回击中特定目标点时的正确发射角度。

该函数接收两个参数，分别表示火炮的长度和发射速度，以及一个向量（数组）参数，即相对于火炮基准的目标位置。该函数计算击中目标点时的火炮的最佳瞄准角度，并返回该角度值（角度值或弧度值）。若火炮未击中目标点，函数应可返回错误消息。对此，可能存在多个有效发射角，读者可任选其一。

【练习 7.3】 试编写 fireCannon(massOfBall, massOfCannon, energy)函数，该函数返回炮弹的离膛发射速度。

该函数通过动量守恒定律计算发射后炮弹的速度和火炮的速度。这里，假设标量 energy 表示发射后的全部动能。也就是说，源自火药的化学能未损失任何热能。

7.6 本章小结

本章快速回顾了弹道学的基本内容，与向量和代数运算相比，相关概念并不复杂。对应求解方法相对固定，多数问题仅是同一主题的不同版本而已。

第 8 章和第 9 章将讨论碰撞检测问题，并再次考察动量守恒和能量守恒定律，以及碰撞过程中的计算方案。

至此，读者应掌握如下内容：

- 加速度和速度变化率。
- 作为恒定向下加速度，近海平面基准上的重力与对象间的作用方式。
- 如何使用运动方程计算弹道问题中的未知项。
- 质量和惯性的含义。
- 动量的含义及其守恒定律。
- 能量的不同形式及其转换方式。
- 如何利用能量守恒定律求解弹道问题。

第 8 章　简单形状之间的碰撞检测

本章包含如下内容：

- 概述。
- 基本原则。
- 圆形间的碰撞。
- 正方形间的碰撞。
- 椭圆形间的碰撞。
- 不同形状对象间的碰撞。

8.1　概　述

碰撞检测及其处理方案可视为游戏程序设计中的基本问题，并涉及大量的数学计算，前述章节亦对此有所提及。本章将对此予以深入考察，并将相关数学知识及其扩展内容引入至物理学范畴中。本章首先介绍 2D 碰撞检测，并为第 3 部分中的旋转物理学打下坚实的基础。第 4 部分则将相关问题扩展至 3D 环境中。

本章将考察两个简单几何形状是否碰撞、何时碰撞，以及如何确定对应碰撞点，因而本章伪代码示例具有一定的特殊性，旨在强调其数学计算过程。与商业应用程序相比，本章示例稍显冗长且缺乏应有的高效特征，其目的仅在于降低读者的理解难度。尽管某些方案提供了优化方案，但这并非是本章的重点内容。

后续章节将深入讨论更为复杂的碰撞环境，为了防止问题过于复杂，并确保读者理解相关概念，本章首先介绍二维碰撞环境。

8.2　基 本 原 则

为了构建基本原则，假设计算面向一个或多个运动对象进行。一般地，针对某一既定对象，当前位置和时间步内的移动向量为已知项。如前所述，这可得到该对象的位移。例如，若对象的移动速度为 **v** 像素/秒，则 150 毫秒后，该对象的位移为 $\frac{100}{50}\times\mathbf{v}$ 像素。

针对更为复杂的运动状态，例如对象处于加速状态，读者依然可根据当前速度执行相同计算。此处，假设碰撞检测较为频繁，且各时间步内速度的变化幅度较小。若速度在当前时间间隔内保

持恒定，则计算结果将不会出现明显的错误。

另一个较为重要的原则是相对性原理，从名称上看，该原理与爱因斯坦的相对论有几分类似。相对性原理表明，若向粒子的物理系统加入恒定速度或位置，则其他测量数据亦须保持一致，该测量隔离方式称作参考坐标系。

为了深入考察参考坐标系，此处考察下列情形：Sam 和 Ella 对坐在厢式货车内并互掷一个球体。在投球的过程中，无论车辆处于静止状态，抑或以 100 公里/小时速度行进，Sam 和 Ella 投球的力度以及球体相对于二人的运动轨迹均未发生变化，这构成了二者的参考坐标系。当然，相对于地面，球体的运动轨迹截然不同，但地面并非是 Sam 和 Ella 参考坐标系中的内容。对此，一类相对极端的例子是遮挡车辆的全部窗户，此时，Sam 和 Ella 甚至无法感受到车辆处于运动状态，抑或处于静止状态，“运动”以及“静止”对二人来讲毫无意义。

【提示】 爱因斯坦的相对论探讨了光的本质，该理论表明，根据相对性原理以及人类的移动速度，时间以不同的速率流逝。爱因斯坦制订了相关运动方程，并消除了与光线行为相关的异常现象，其观点颠覆了大多数人对空间和时间的理解。当前，大量的观察结果和试验均验证了爱因斯坦这一观点的正确性。

为了简化碰撞实体的讨论过程，本章和后续章节采用了面向对象编程风格。相应地，各碰撞实体可定义为一个对象。总体而言，各对象均包含相应特征，并采用对应属性加以描述。这里，假设某一对象名为 circle，其属性之一为 radius，另一个属性为 circumference。当访问其中的一个属性时，可采用“ • ”操作符进行操作。例如，若将 circle 对象的 radius 属性赋值为 20，对应代码可写为 circle • radius = 20。当采用此类语法规则时，则无须编写包含多个参数的函数。附录 B 将对面向对象程序设计进行了简要的回顾。

8.3　圆形对象间的碰撞

本节介绍一类最为简单的碰撞检测形式，即圆形对象间的碰撞。其简单之处在于，圆形呈对称状态，因而圆周上各点具有某种共性，这将对碰撞计算起到简化作用。

8.3.1　圆形

前述章节曾对圆形有所提及，圆形可视为与特定点等距的点集，该特定点表示为圆心 O。圆周至中心位置之间的距离则称作半径。针对某一角度 θ，O 与圆周上任意一点 P 之间的向量记作 $r(\sin\theta\ \cos\theta)^{T}$。同时，点 P 处的圆切线垂直于半径 $\overline{OP}$。这里，可方便地将半径为 r、位置向量为 $\mathbf{o}$（圆心位置）的圆记为 $C(\mathbf{o}, r)$。

另外，可方便地计算点 P 是否位于圆内部，对此，仅需计算 OP 的距离，并将结果值与半径进行比较。若该距离值小于半径，则点 P 位于圆内部；若大于半径值，则该点位于圆外部；若距离值等于半径，则该点位于圆周上。

【提示】相应地，可计算平方距离值 OP^2，并与半径平方值进行比较，进而加速计算过程，这将避免基于毕达哥拉斯定理的平方根计算。

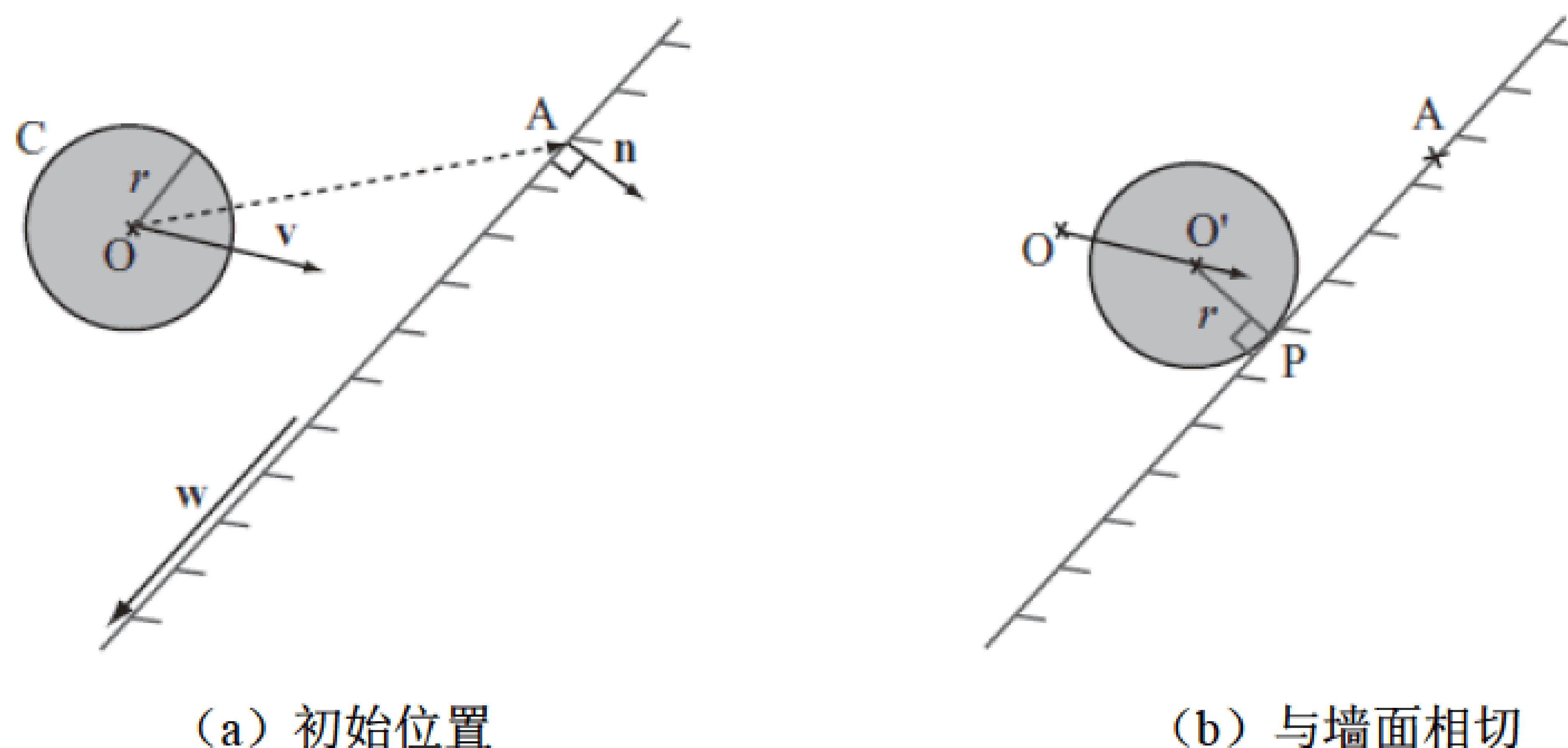
若圆心为 O 和 O'的两个圆于点 P 处相切，则点 O、P 和 O'彼此共线，该结论源自下列事实：位于点 P 处的切线垂直于 $\overrightarrow{OP}$ 和 $\overrightarrow{O'P}$，这一关系有助于读者从几何角度求解碰撞问题。

8.3.2 移动的圆形和墙壁

图 8.1（a）显示了处于运动状态的圆形和墙壁之间的碰撞关系。其中，圆 C(**o**, *r*)以速度 **v** 向一条直线行进。

在图 8.1（a）中，若给定时间步，则已知项包括：时刻 0 时圆 C 的圆心位置 O，直线上的点 A（其位置向量为 **a**），以及沿该直线的向量 **w**（在当前示例中，假设墙面无限长）。若给定 **v**，**w**，**o**，**a**，**r** 的相关值，则圆是否会在当前时间步内与墙面碰撞？若尝试碰撞，该碰撞行为何时出现？

假设圆形对象与墙面发生碰撞，如图 8.1（b）所示。其中，圆 C 于点 P 处与墙面接触，且对应圆心位置为 O'。由于墙面与圆 C 相切于点 P，因而 $\overrightarrow{O'P}$ 垂直于 **w**。另外，由于 $\overrightarrow{OO'}$ 同为圆 C 的运动方向，因而该向量表示为 **v** 的倍数。



（a）初始位置　　（b）与墙面相切

图 8.1　圆形对象朝向墙面运动

读者是否可根据上述信息得出相关结论，并定义碰撞检测函数？对此，可首先考察 $\overrightarrow{O'P}$（此处暂且将其称作 **r**）。此处，**r** 垂直于 **w**，且长度值表示为 *r*。相应地，可通过向量 $\overrightarrow{OA}$ 计算其方向。具体而言，可使用 **w** 的法线 **n** 予以计算。若 **n** 和 $\overrightarrow{OA}$ 之间的角度大于 90°，则 **n** 指向 O 且有 **r** = *r***n**；若该角度小于 90°，则 **n** 背向当前圆形对象且有 **r** = –*r***n**。为了区分这两种情形，可计算 **n** 和 $\overrightarrow{OA}$ 之间的点积结果。这可确定 **n** 方向上的 $\overrightarrow{OQ}$ 分量，若该分量的绝对值小于 *r*，则圆内嵌于墙面内，此时对应函数应返回错误消息。

待 **n** 计算完毕后，还可执行另一项快速检测。若 **v** 和 **n** 的点积值为正数，则圆形对象背离墙面运动；否则，则可得到一个简单的运动方程。当前，P 位于 AB 上，且向量 **r** 背离 O 上的轨迹点。针对标量 *s* 和 *t*，可得到如下算式：

$$\mathbf{o} + t\mathbf{v} + \mathbf{r} = \mathbf{a} + s\mathbf{w}$$

一旦推导出上述方程，则可定义 circleWallCollision()函数，进而检测圆和墙面之间的碰撞效果。下列内容显示了该函数的伪代码，需要注意的是，代码采用了面向对象语法，进而标识圆对象属性以及墙面对象，如下所示：

```
function circleWallCollision(cir, wal)
//calculate the normal to the wall
set n to wal.normal
set a to wal.startPoint-cir.pos
set c to dotProduct(a, n)
if abs(c)<cir.radius then return "embedded"
if c<0 then set r to n*radius
otherwise set r to -n*radius

//check if the circle is approaching the wall
set v to dotProduct(displacement, n)
if v>0 then return "none"

//calculate the vector equation
set p to cir.pos+r
set t to intersectionTime(cir.pos,cir.displacement,wal.startPoint,wal.vector)
//see Chapter 5
if t>1 then return "none"
return t
end function
```

函数 circleWallCollision()使用了第 5 章讨论的函数。然而，关于 intersectionTime()函数的调用，须提供 4 个参数：p1，v1，p2，v2。该函数根据 p1 返回 t 和 v1 之间的比例，即在与直线（源自 p2 且沿向量 v2）相交之前行进的距离。在第 5 章中曾讨论到，关于该值的交点，读者可尝试绘制对应的计算结果。例如，若 t 位于[0,1]区间内，则交点出现于目标时间段内。针对当前参考坐标系，在圆与墙面的靠近过程中，t 值总保持大于等于 0 的状态，该结论源于当前示例，并多次出现在后续章节中。

8.3.3　静止圆和运动点

另一个与圆形有关的碰撞示例则涉及较小对象，例如空间中移动的点粒子。对此，需要确定点与圆是否发生碰撞（在该圆占据的同一空间内）。

如前所述，判断一点是否位于圆内部并不复杂，读者需要确定粒子何时进入圆半径范围内。在图 8.2 中，除了位置 P 和向量 **v** 之外，位置 O 和圆半径 r 亦为已知项。

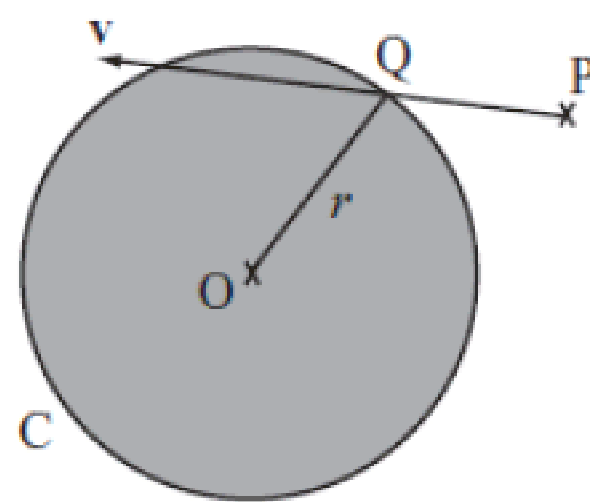


图 8.2　静止圆和运动点

为了求解点与圆之间的相对位置，可考察图 8.2 中的点 Q，其中，粒子开始进入圆内部区域。这里，点 Q 位于圆 C 上，同时也位于粒子的运动轨迹上。针对某一 t 值，其位置向量可表示为 $\mathbf{p} + t\mathbf{v}$，该点至 O 之间的向量值表示为 r。据此，可推导出下列向量方程：

$$(\mathbf{p} + t\mathbf{v} - \mathbf{o}) \cdot (\mathbf{p} + t\mathbf{v} - \mathbf{o}) = r^2$$

除了 t 值以外，全部数据均为已知项，因而该方程易于求解。待执行括号展开以及乘法运算后，对应结果如下所示（其中，$\mathbf{w}$ 表示为向量 $\mathbf{p} - \mathbf{o}$）：

$$\mathbf{w}\cdot\mathbf{w}+2t\mathbf{w}\cdot\mathbf{v}+t^2\mathbf{v}\cdot\mathbf{v}=r^2$$

$$t=\frac{-\mathbf{w}\cdot\mathbf{v}\pm\sqrt{(\mathbf{w}\cdot\mathbf{v})^2-(\mathbf{w}\cdot\mathbf{w}-r^2)(\mathbf{v}\cdot\mathbf{v})}}{|\mathbf{v}\cdot\mathbf{v}|}$$

尽管上式稍显复杂，但依然源自简单的计算过程。最终结果可描述为，若 t 小于 0，则当前点背离 C 运动；若 t 大于 1，则该点在时间步内未与 C 相交；若 t 为虚数（根号中的数据小于 0），则该点的运行轨迹未与圆相交。pointCircleCollision()即采用这一计算过程，如下所示：

```
function pointCircleCollision(pt, cir)
  set w to pt.pos-cir.pos
  set ww to dotProduct(w,w)
  if ww<cir.radius*cir.radius then return "inside"
  set v to pt.displacement-cir.displacement
  set a to dotProduct(v,v)
  set b to dotProduct(w,v)
  set c to ww-cir.radius * cir.radius
  set root to b*b - a*c
  if root<0 then return "none"
  set t to (-b-sqrt(root))/a
  if t>1 or t<0 then return "none"
  return t
end function
```

根据相对性原理，圆-直线以及圆-点位置关系问题具有“双向”特征。根据 circleWallCollision()函数，可将圆 C 视为静止且墙面以速度$-\mathbf{v}$ 运动，而非圆以速度 $\mathbf{v}$ 向墙面移动。对此，墙面首先与圆 C 上的点 Q 相交，且点 Q 处的切线与墙面平行。若从点 Q 沿速度向量绘制一条直线，则可计算点 P。该问题与 pointCircleCollision()函数类似，后者已知点 P 且计算点 Q，而前者则已知点 Q 并计算点 P。

8.3.4　直线上的两个运动圆

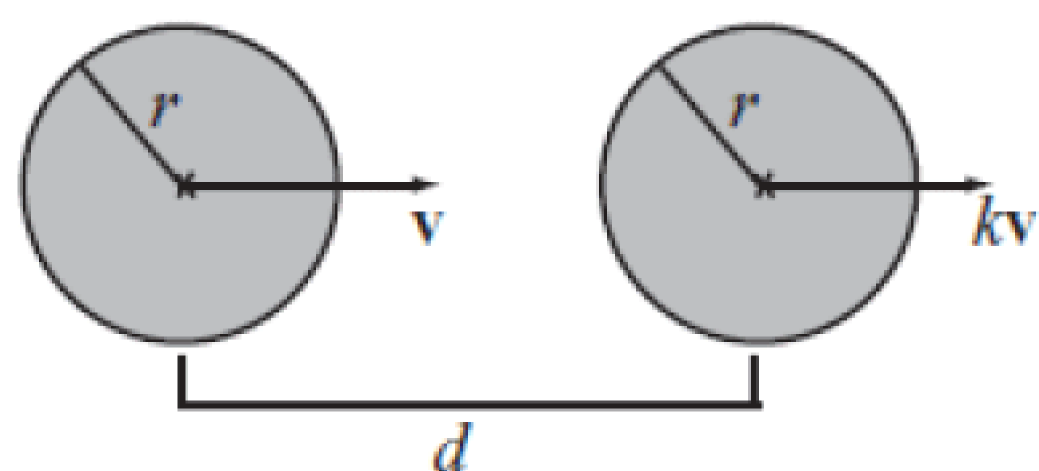
假设两个球体对象位于铁轨上且相距一段距离，若向球体对象 2 滚动对象 1，则二者何时碰撞？若两个对象皆处于运动状态，情况又当如何？如图 8.3（a）所示，两个球体对象的半径为 r，相距 d 个单位且速度分别为 $\mathbf{v}$ 和 $k\mathbf{v}$。相应地，两个球体沿同一直线运动，因而对应速度也将彼此平行。

若两个球体皆处于运动状态，根据相对性原理，一个球体处于运动状态（另一个球体处于静止状态），或两个球体皆处于运动状态，二者间的计算结果并无差别——具体而言，可从两个球体速度中减去球体 1 的速度，即球体 1 处于静止，而球体 2 以$(k–1)\mathbf{v}$ 速度运动。

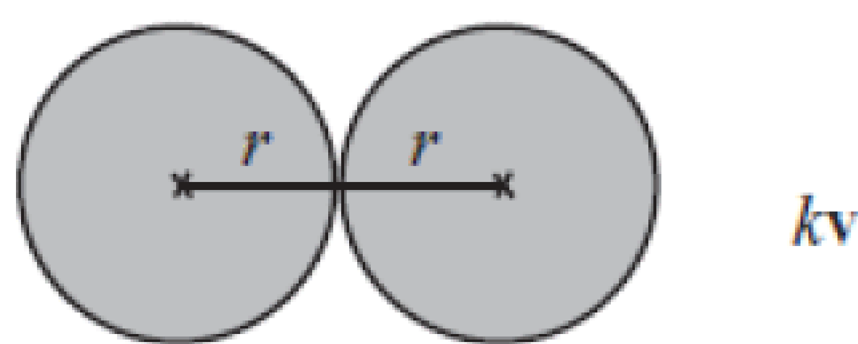
在图 8.3（b）中，当球体间彼此碰撞时，球心之间相距 $2r$ 个单位。这意味着，全部工作将通过标准的运动方程以及 $2r$ 距离计算 t 值，下列算式显示了其中的一种方案：

$$|\mathbf{v}|t = d-2r$$

$$t=\frac{d-2r}{|\mathbf{v}|}$$



（a）碰撞前



（b）碰撞过程

图 8.3　直线上的运动圆

若圆心位于碰撞直线上，则可对上述方案进行适当扩展，并包含不同尺寸的圆。据此，读者可尝试编写 circleCircleStraightCollision()函数，如下所示：

```
function circleCircleStraightCollision(cir1, cir2)
  set relspeed to cir1.speed-cir2.speed
  set d to cir1.pos-cir2.pos //linear position
  set r to cir1.radius+cir2.radius
  if d<r then return "embedded"
  set t to (d-r)/relspeed
  if t>1 or t<0 then return "none"
  return t
end function
```

circleCircleStraightCollision()函数并未定义速度和位置向量，相反，该函数使用了速率和线性位置，并适用于下列情形：预先知晓两个圆将发生迎面碰撞。此时，速度向量定位于两个圆心的直线上。尽管如此，为了确定碰撞点，此处依然需要计算速度向量。

8.3.5　以某一角度运动的两个圆

前述示例可处理正面碰撞的两个对象，而多数时候，对象间常以某一角度发生碰撞。对此，需要一种通用方案处理不同角度、不同尺寸的碰撞圆。该方案貌似困难，实际上，前述内容已完成了大部分工作。

如图 8.4 所示，圆 D($\mathbf{a}$, p)以相对速度 $\mathbf{v}$ 靠近圆 C($\mathbf{o}$, r)，此处，可围绕 O 绘制一个半径为 $p + r$ 的较大圆。当两个圆彼此碰撞时，直线（源自 A 且方向为 $\mathbf{v}$）进入较大圆中。

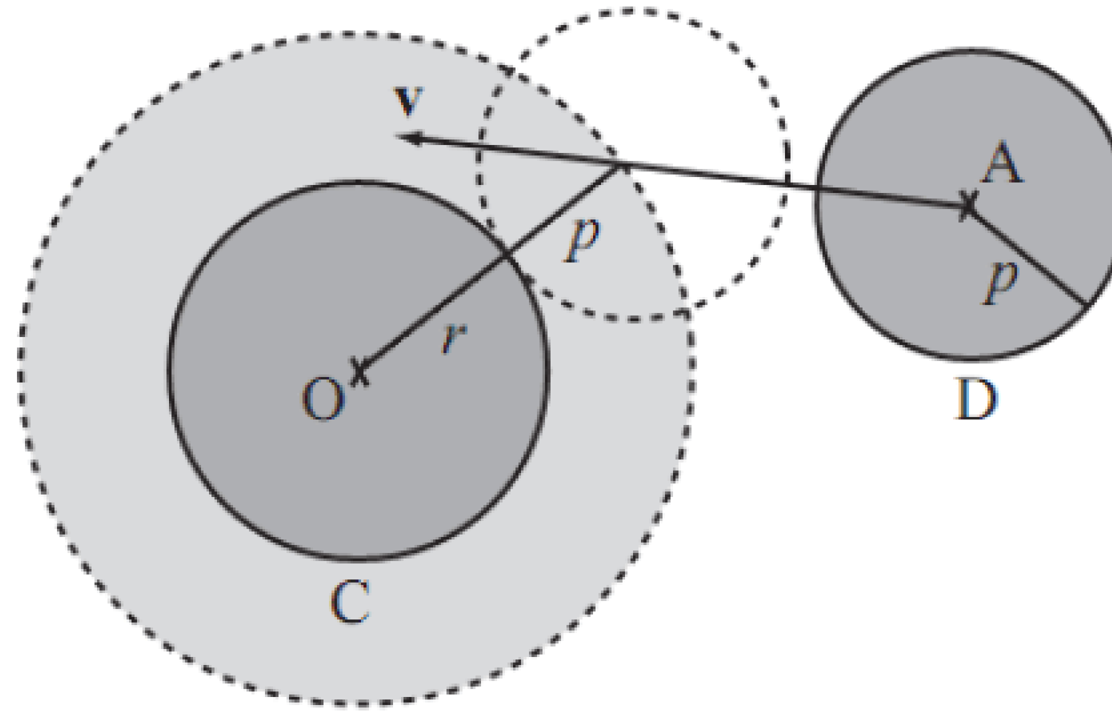


图 8.4　以某一角度运动的两个圆

当给定上述信息后，则可重写 pointCircleCollision()函数，进而处理以某一角度运动的两个圆，对应函数为 circleCircleCollision()，如下所示：

```
function circleCircleCollision(cir1, cir2)
  set w to cir1.pos-cir2.pos
  set r to cir1.radius+cir2.radius
  set ww to dotProduct(w,w)
  if ww<r*r then return "embedded"
  set v to cir1.displacement-cir2.displacement
  set a to dotProduct(v,v)
  set b to dotProduct(w,v)
  set c to ww-r*r
  set root to b*b-a*c
  if root<0 then return "none"
  set t to (-b-sqrt(root))/a
  if t>1 or t<0 then return "none"
  return t
end function
```

8.3.6　内嵌圆

前述内容讨论了外部区域中的圆对象碰撞行为，然而，图 8.5 显示了另外一种情形，即一个圆形对象在另一个较大的圆中运动。

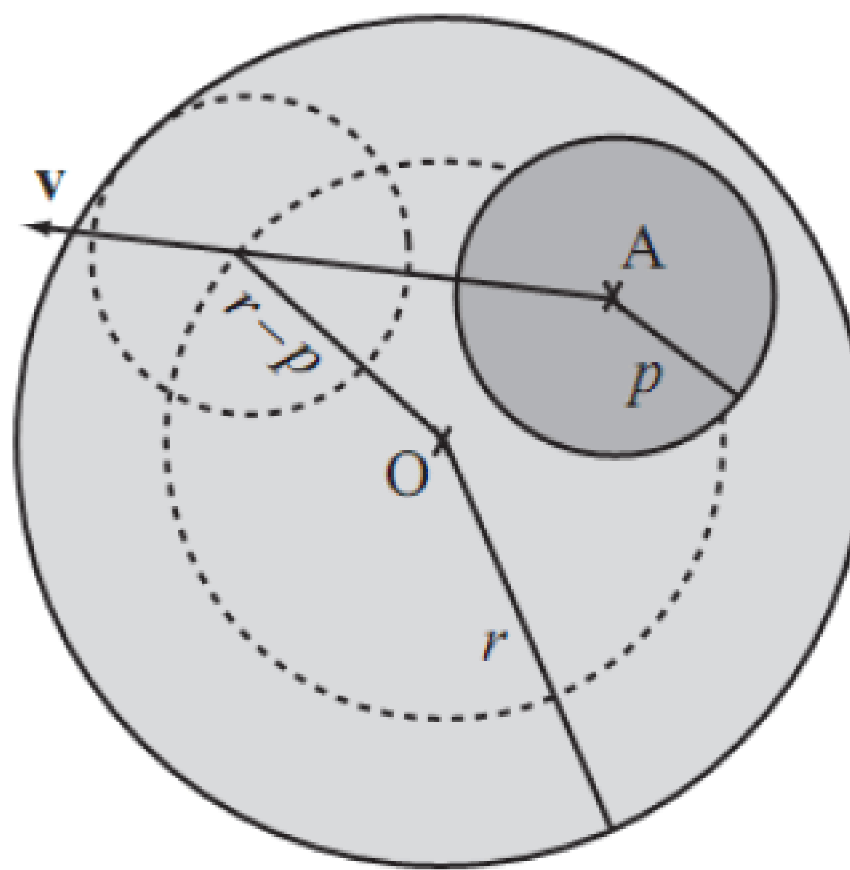


图 8.5　一个圆内嵌于另一个圆中

图 8.5 中所示类似于一种内部碰撞，当两个圆彼此碰撞时，圆心彼此相距某一段距离 A。即两个圆的半径差。相比较而言，当发生外部碰撞时，圆心间的距离为半径之和。

相关函数类似于 circleCircleCollision()函数，由于当前示例计算半径差（而非半径和），因而 circleCircleInnerCollision()函数不再对称。需要注意的是，当前示例测试圆 1 是否位于圆 2 内部，而非相反。另外，此处还需计算较大根值。若两个圆处于内嵌状态，则速度向量将分前后两次与圆发生碰撞，对应代码如下所示：

```
function circleCircleInnerCollision(cir1, cir2)
  set w to cir1.pos-cir2.pos
  set r to cir2.radius-cir1.radius
  set ww to dotProduct(w,w)
  if ww>r*r then
    set rr to cir2.radius+cir2.radius
    if ww<rr*rr then return "embedded"
    return "outside"
  end if
  set v to cir1.displacement-cir2.displacement
  set a to dotProduct(v,v)
  set b to dotProduct(w,v)
  set c to ww-r*r
  set root to b*b-a*c
  set t to (-b+sqrt(root))/a
  if t>1 then return "none"
  return t
end function
```

8.3.7　碰撞点

圆与另一对象之间的碰撞点计算涉及诸多内容。首先，读者可根据前述章节中的函数结果快速计算碰撞点位置，其中，各函数均返回变量 t，该值表示为对象碰撞前的时间值。若对象的速度已知（通常情况下，速度为已知项），则可计算 shape.pos+shape.displacement * t，进而获取当前最新位置。

其次，读者还可尝试计算实际碰撞点，尽管碰撞过程处于变化状态中。如图 8.1 所示，当圆形对象与墙面碰撞时，碰撞点（P）位于圆心与墙面之间的垂线上。另外一方面，如图 8.4 所示，当两个圆碰撞时，碰撞点位于连接圆心的直线上，即图 8.2 中的 $\overrightarrow{OQ}$。位于该点的圆切线垂直于半径，这一结论对于后续讨论十分重要。

8.4　正方形碰撞

除了平滑形状之外，本小节还将考察包含边角的一类形状，例如正方形和矩形。类似于圆形，

此类形状包含诸多有用属性，其对称性也大大降低了计算难度。另外，直线碰撞检测可视为大多数复杂碰撞的基础内容。

8.4.1 正方形和矩形

尽管前述章节曾对正方形和矩形有所提及，但本节将再次回顾某些细节内容。矩形可视为包含直边的二维形状，通常也称作多边形。矩形包含 4 个顶点并构成了一个四边形。同时，矩形包含 4 个直角且对边相等，彼此平行，因而也可视为一个平行四边形。针对矩形 ABCD，顶点采用顺时针方式标记，且有 $\overrightarrow{AB}=\overrightarrow{DC}$，$\overrightarrow{BC}=\overrightarrow{AD}$；同时，$\overrightarrow{AB}$ 垂直于 $\overrightarrow{BC}$。正方形可视为矩形的一个特例，其中，AB 和 BC 等长，因而正方形的全部边长均相等。

对角线 $\overrightarrow{AC}$ 和 $\overrightarrow{BD}$ 相交于矩形 ABCD 的中心位置，在正方形对角线之间彼此互相垂直。另外，对角线 $\overrightarrow{AC}$ 等于 $\overrightarrow{AB}+\overrightarrow{BC}$，对角线 $\overrightarrow{BD}$ 等于 $\overrightarrow{AD}-\overrightarrow{AB}$，该结论对于任意平行四边形均成立，这也是为何向量加法常称作平行四边形法则的原因。

【提示】 下面对四边形的分类方式稍作扩展：菱形表示为各边均等的平行四边形；梯形为两边平行的四边形；而在等腰梯形中，两条非平行边等长；风筝形状同样为平行四边形，并包含两组相等的邻边。这里，正方形也是一类矩形和菱形，且二者皆为平行四边形；菱形也可视为一类风筝形图元。同样，平行四边形也是一个梯形。上述全部图形皆为四边形（凸四边形）。

读者可尝试使用基本符号定义矩形。例如，在矩形 R(**u**, **v**, **w**)中，中心点为位置向量 **u**，各边由垂直向量 2**v** 和 2**w** 确定，其中，|**v**| > |**w**|。虽然还存在简单方案（参见前述章节），当前矩形定义方案体现了一种对称特征。除此之外，若 **w** 可赋予任意值（而非垂直于 **v** 的向量），则可得到一类更为通用的四边形。

严格地讲，上述矩形描述包含了某些冗余信息。由于两边的垂直特征即可确定第 2 条边的方向，因而可定义一个边向量和另一边的长度（正标量值），该方案可消除潜在错误。相应地，3 个数据值的任意组合即可生成一个有效的矩形。针对退化现象（一条边为 0），对应结果为一条直线段而非矩形。如前所述，第一种方案更为简单且具有对称特征。

当采用 R(**u**, **v**, **w**)这一形式时，可方便地确定矩形 R 的顶点，即 **u** + (**v** + **w**)，**u** + (**v** – **w**)，**u** – (**v** + **w**)，**u** – (**v** – **w**)。需要注意的是，本章以及后续章节均采用这一顶点顺序。另外，矩形也可具有一定的方向，即沿其较长一边的方向，因此，矩形 R 朝向向量 **v**。

如图 8.6 所示，当编写函数以测试点 P 是否位于矩形 R 中时，可计算 **v** 和 **w** 方向上向量 $\overrightarrow{UP}$ 的分量。若方向 **v** 的分量小于|**v**|，且方向 w 的分量小于|**w**|，则点 P 位于矩形 ABCD 内。

函数 pointInsideRectangle()使用了如下算法：

```
function pointInsideRectangle(pt, rectCenter, side1, side2)
  set vect to pt-rectCenter
  set c1 to abs(component(vect, side1))
  set c2 to abs(component(vect, side2))
```

```
  if c1 > magnitude(side1) then return false
  if c2 > magnitude(side2) then return false
  return true
end function
```

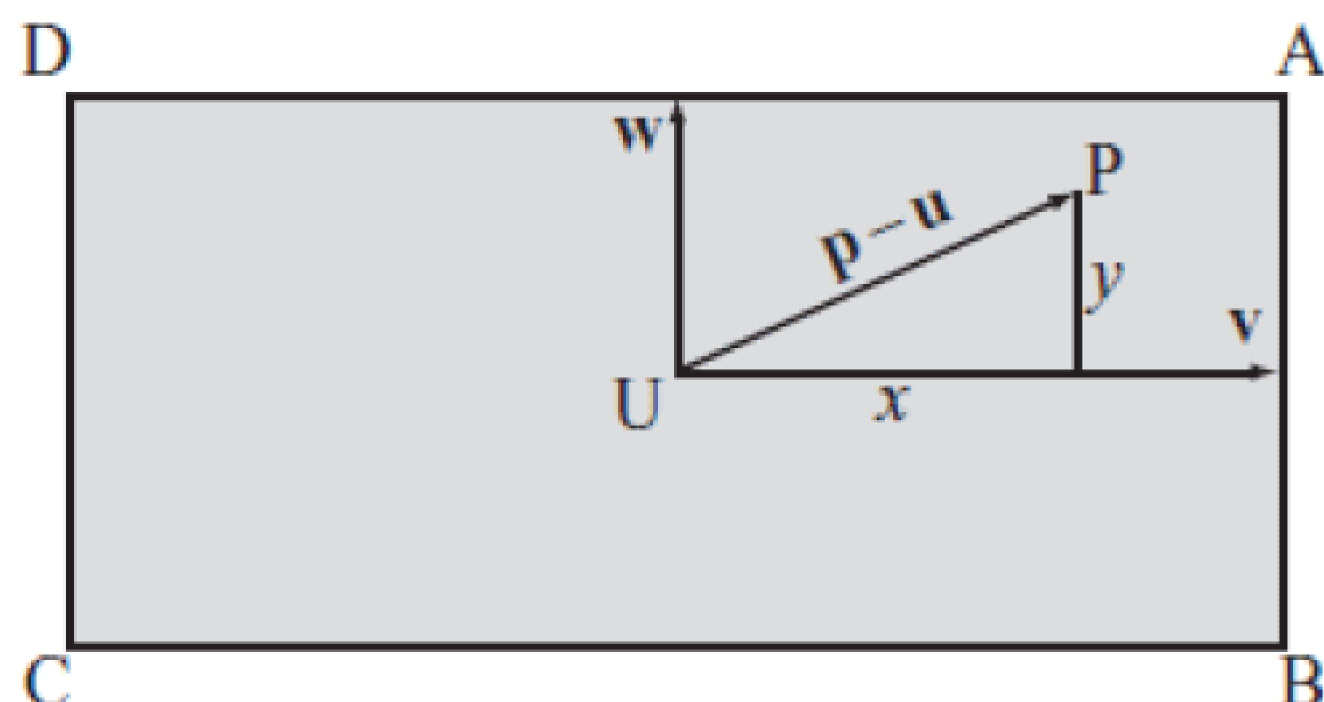


图 8.6　确定一点是否位于矩形内部

第 9 章将采用另一种通用方法测试一点是否位于多边形内部。当前需要注意的是，pointInside Rectangle()函数将位于周长上的点视为内部点。当然，读者可将“>”号替换为“>=”号，进而对原规则进行修改。

为了定义相对完善的函数，针对矩形 R 周长上的点 P，若采用向量 $\overrightarrow{UP}$，则 **v** 方向上的分量大小为|**v**|，抑或 **w** 方向上的分量大小为|**w**|。若二者皆为真，则点 P 为矩形 R 的顶点。相反，若满足某一条件以及上述“点位于 R 内部”之定义，则点 P 位于矩形周长上。针对任意平行四边形，pointOnRectangle()函数即采用了上述各结论，如下所示：

```
function pointOnRectangle(pt, rectCenter, side1, side2)
  set vect to pt-rectCenter
  set c1 to abs(component(vect, side1))
  set c2 to abs(component(vect, side2))
  set s1 to magnitude(side1)
  set s2 to magnitude(side2)
  if c1 > s1 then return false
  if c2 > s2 then return false
  if c1=s1 or c2=s2 then return true
  //NB: for a safer test, use e.g. abs(c1-s1)<0.001
return false
end function
```

8.4.2　静止矩形和运动点

这里可对 8.4.1 节内容稍作扩展，并分析静止矩形和运动点之间的碰撞检测。为了求解该问题，下面考察一类简单的情形：粒子位于点 P 处，并以速度 **v** 穿越某一平面，该平面包含矩形 R(**u**, **a**, **b**)，如图 8.7 所示。

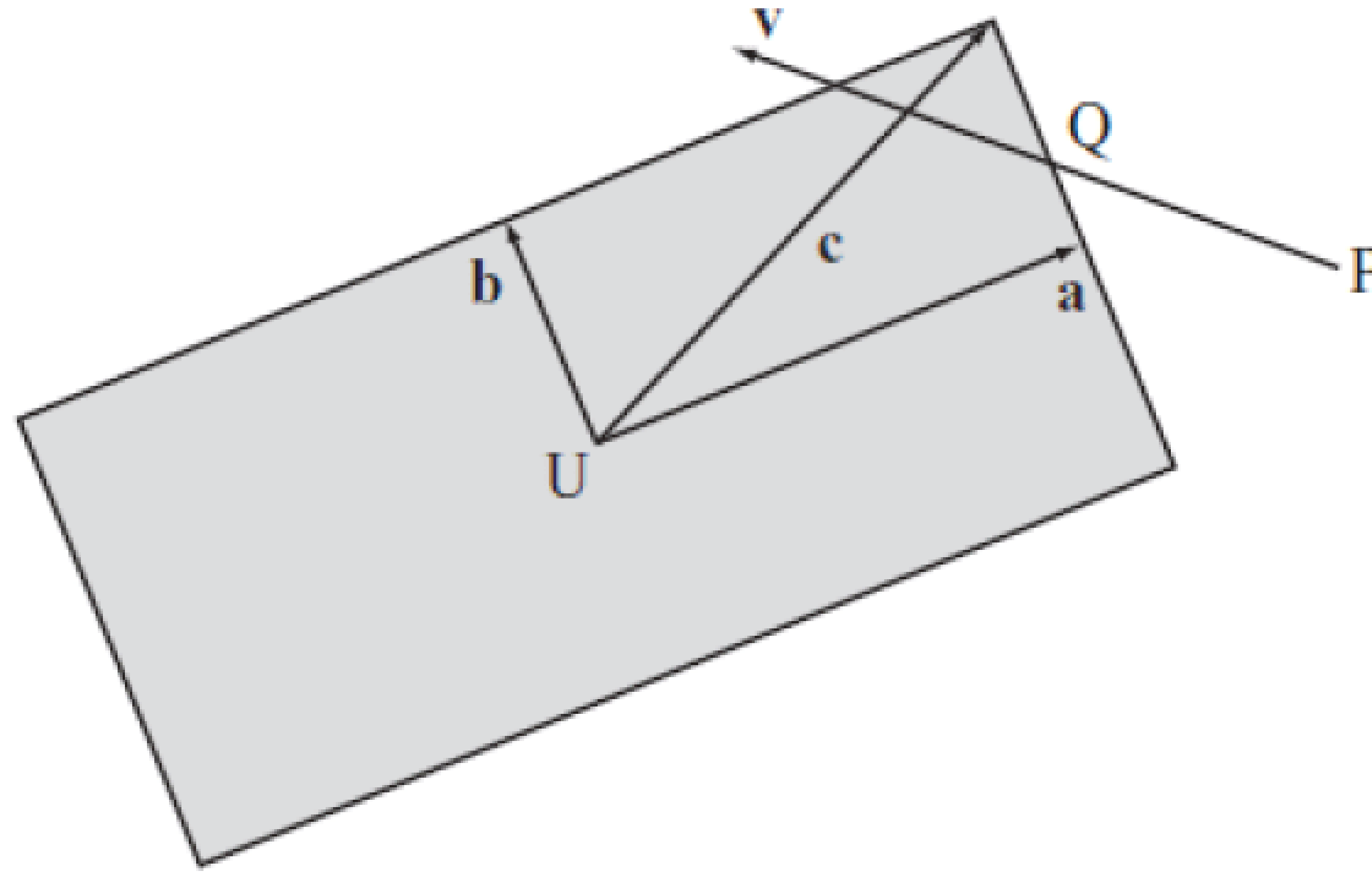


图 8.7　静止矩形和运动点

此处需要计算点 Q，以使 Q 位于粒子的运动轨迹以及 R 的边界上。对此，需要测试各条边的交点。在首个交点处，粒子与矩形发生碰撞。pointRectangleIntersection()函数体现了对应的实现方式，如下所示：

```
function pointRectangleIntersection(pt, rec)
  set c to rec.side1+rec.side2
  set t to 2 //start with a high value of t
  //then repeat over the four sides and look for the first collision
  repeat for v = rec.side1, rec.side2
    repeat for m = 1,-1
      set t1 to intersectionV(pt.pos,pt.displacement,rect.pos-m * c,  m * v *
                             rec.axis*2)
      if t1 = "none" then next repeat
      set t to min(t,t1[1])
    end repeat
  end repeat
  if t=2 then return "none"
  return t
end
```

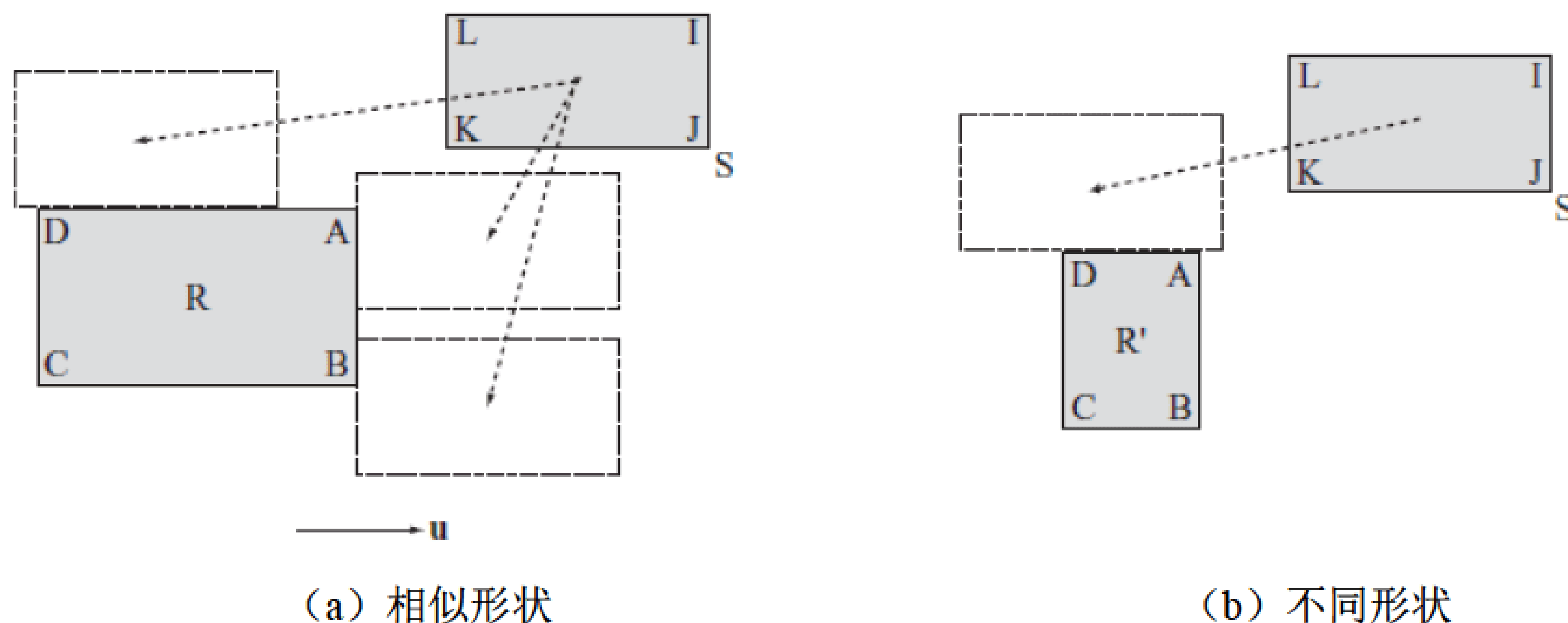
pointRectangleIntersection()函数遍历 4 条边并构造一个向量。此处将生成 $\mathbf{c} = \mathbf{a} + \mathbf{b}$，即 R 中心位置至某一顶点之间的向量。该构造过程表明，$-\mathbf{c}$ 表示为指向对顶点的向量。根据第一个顶点，两条边分别指向$-\mathbf{a}$ 和$-\mathbf{b}$ 方向；根据第二个顶点，其他两条边则分别指向 $\mathbf{a}$ 和 $\mathbf{b}$ 方向。

pointRectangleIntersection()函数调用 intersectionV()函数，后者可视为 intersection()函数的变化版本，并使用位置-位移信息（而非 4 个端点）。随后，若两条直线段相交，该函数则返回基于相交时刻的 t 值。

由于尚不存在连续的数学函数可描述矩形上的全部点，因而需考察与矩形碰撞检测相关的多种方案，即检测多种可能方案，并谨慎处理顶点处的潜在问题，在 pointRectangleIntersection()函数示例中，问题源自点 P 是否位于矩形的延长边上，并平行于该边运动。当前函数是否可捕捉到这一移动方式？对此，读者可参考练习 8.1 并尝试处理此类问题。

8.4.3　同一角度碰撞的矩形

除了粒子和矩形之间的碰撞检测之外，下一个问题是两个矩形之间的碰撞行为。图 8.8 显示了一类较为简单的示例，其中，两个矩形沿同一轴对齐。



（a）相似形状　　（b）不同形状

图 8.8　同一角度的两个矩形

图 8.8 显示了两个矩形 R 和 S，且皆与轴 **u** 对齐。在图 8.8（a）中，取决于速度向量，矩形 S 可通过多种方式与矩形 R 发生碰撞，并归结为 6 种情况。在当前示例中，碰撞点处至少 3 个顶点（即 S 的 J，K，L）中的一个顶点与 R 的一条边碰撞，该边为 R 中的 AB 或 AD 边。

进一步讲，当 K 与 AB 或 AD 碰撞时，J 定位于 AD 上（或 AB 和 AD），L 定位于 AB 上（或 AB 和 AD）。其他 3 种情况源自图 8.8 所示场景，其中，沿碰撞边的矩形 R'小于 S。因此，在碰撞点处，不存在 S 的顶点与 R'碰撞。然而，R'的顶点与 S 碰撞。这也意味着，可从相反方向执行类似的计算集。在其他方向上，读者只需针对顶点 B 和 D 检测碰撞行为。

作为 pointRectangleCollision()函数（该函数测试全部 4 条边的相交结果）的改写版本，rectangleRectangleCollisionStraight()函数并非使用到前述内容所提及的优化方案。当然，读者可尝试实现同一测试的快速处理方案。对此，一种方法是对 pointRectangleCollision()函数进行适当调整。下列代码显示了两个矩形之间的碰撞处理过程：

```
function rectangleRectangleCollisionStraight(rec1, rec2)
  set t1 to rrVertexCollisionStraight(rec1, rec2)
  set t2 to rrVertexCollisionStraight(rec2, rec1)
  if t1="none" then return t2
  if t2="none" then return t1
  return min(t1,t2)
end function
```

为了对 rectangleRectangleCollisionStraight()函数提供有效的支持，这里需要调用 rrVertexCollisionStraight()函数两次，进而对碰撞行为予以检测，该函数如下所示：

```
function rrVertexCollisionStraight(rec1, rec2)
  set xvector to rec1.axis
```

```
    set yvector to normal(rec1)
    set r1 to rec1.side1*xvector
    set r2 to rec1.side2*yvector
    //calculate the points to test
    set points to pointsToCheck(r1.pos, r2.pos,r1.displacement - r2.displacement)

    //now test each of these for intersection with the second rectangle
    set s1 to rec2.side1*xvector
    set s2 to rec2.side2*yvector
    set t to 2 //you're trying to find a value less than 1 for t
    repeat for each pt in points
      set t2 to pointRectangleIntersection(pt, rec)
      if t2="none" then next repeat
      set t to min(t, t2)
    end repeat
    if t=2 then return "none"
    return t
  end function
```

如前所述，可通过 pointsToCheck()函数检测位置和位移，如下所示：

```
  function pointsToCheck(r1, r2, displacement)
    set points to an empty array
    set c1 to component(displacement, r1)
    set c2 to component(displacement, r2)
    if c1>0 then
      add r1+r2 to points
      add r1-r2 to points
    otherwise
      add -r1+r2 to points
      add -r1-r2 to points
    end if
    if c2>0 then
      if c1>0 then add -r1+r2 to points
      otherwise add r1+r2 to points
    otherwise
      if c1>0 then add -r1-r2 to points
      otherwise add r1-r2 to points
    end if
  end function
```

针对同一角度的矩形碰撞检测，编写 3 个函数似乎稍显复杂，但其工作状况尚且良好。不难发现，其复杂度主要源自需要检测不同的碰撞状态。类似于点和矩形，相关计算同样适用于对齐的平行四边形。与 rrVertexCollisionStraight()函数所采用的方案相比，其差别在于需要确定 yvector 变量值，而非基于 xvector 变量的法线。

若矩形呈轴对齐状态，则可使用基于上述处理方法的一类简化版本。轴对齐矩形在两个基向量方向上呈对齐状态，因而易于处理，多数章节对此均会有所提及。

8.4.4　不同角度的两个矩形

若矩形沿同一轴未处于对齐状态，读者可能会认为问题将趋于复杂化，事实并非如此。实际上，与同一角度的矩形相比，其处理难度甚至有所下降——碰撞点为 8 个顶点中的某一顶点。如图 8.9 所示，读者可尝试将顶点数量降至 6 个，且每 3 个顶点分别来自不同的矩形。

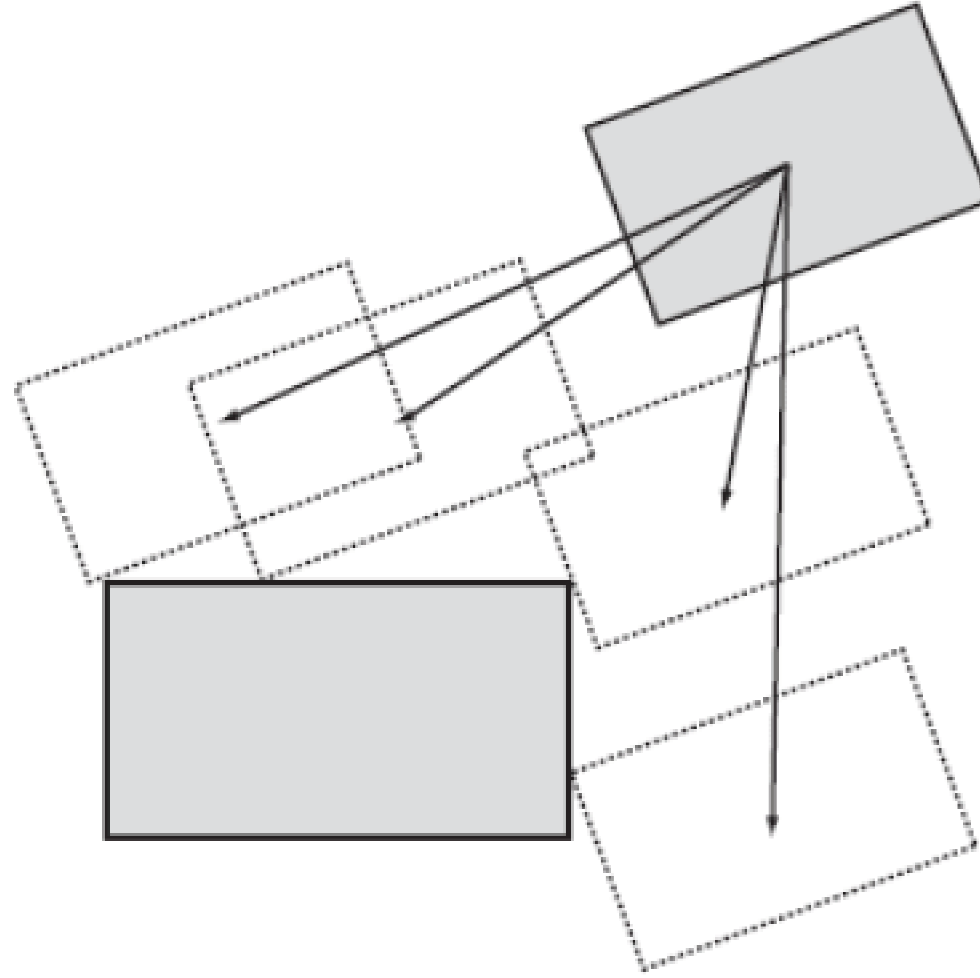


图 8.9　不同角度的矩形

如 rectangleRectangleAngledCollision()函数所示，为了计算需要检测的顶点，读者可将位移向量与矩形的对角线进行比较。考虑到差异性，可将矩形边值作为向量，该方案可使用前述已完成的工作，如下所示：

```
function rectangleRectangleAngledCollision (rec1, rec2)
  set t1 to rrVertexCollisionAngled(rec1, rec2)
  set t2 to rrVertexCollisionAngled(rec2, rec1)
  if t1="none" then return t2
  if t2="none" then return t1
  return min(t1,t2)
end function
```

rrVertexCollisionAngled()函数在 rectangleRectangleAngledCollision()函数中调用两次，其功能可描述为：首次“到达”测试点，并于随后测试交点。

```
function rrVertexCollisionAngled(rec1, rec2)
  //calculate the points to test
  set axis to rec1.axis
  set points to pointsToCheck(rec1.side1* axis,  rec1.side2*normalVector(axis),
                              displacement)
  //now test each of these for intersection with the second rectangle
  set t to 2
  repeat for each pt in points
    set t2 to pointRectangleIntersection(pt, rec2)
    if t2="none" then next repeat
    set t to min(t, t2)
```

```
  end repeat
  return t
end function
```

在上述代码以及前述章节中，对齐矩形与非对齐矩形并未显示出明显的差别，但在轴对齐盒体中，仅须考察一点，相关内容将在 8.4.5 节加以讨论。

8.4.5 碰撞点

平滑对象（例如圆形）和多边形（例如矩形）之间的主要差别在于碰撞点。当两个平滑对象方式碰撞时，通常在碰撞点处存在一条法线，进而可精确地定义碰撞行为。而多边形则与此不同：多边形的两条边之间可产生碰撞，如图 8.8 所示；或者，多边形的边与顶点之间也可能存在碰撞，如图 8.9 所示。

【提示】从理论上讲，也可计算两点之间的碰撞结果。若该现象较为明显，则可采用扰动行为对其进行处理，当前暂不对这一现象进行讨论。

若两条边之间产生碰撞，则碰撞检测过程涉及相交边的法线计算。然而，顶点并不包含法线，当顶点和多边形边产生碰撞时，读者可能认为会出现多种情况。实际上，具体情况并不复杂。当顶点与边相交，至少存在一条定义明确的法线。

对应求解过程相对直观，只需简单地采用既定函数，并返回矩形碰撞点和其他有用信息。

类似地，矩形和墙面之间的碰撞检测几乎等同于两边的碰撞行为。其中，墙面可视为一个较大的矩形，且远远大于与其碰撞的矩形。由于无须测试墙面顶点与矩形顶点之间的碰撞行为，因而当前问题可划分为两部分内容。另外，还可执行与直线段之间的碰撞计算。实际上，直线段可视为某一组边长为 0 的矩形。

8.5 椭圆形之间的碰撞

椭圆（ellipse）可视为一类扁平的椭圆状图案，而椭圆曲线（oval）则表示为一类卵形形状。椭圆的简单性类似于圆，但二者并不完全对等，因而需要对二者区别对待。具体而言，圆形包含无穷多个对称轴，而椭圆仅包含两个对称轴。最终，椭圆与圆形之间的碰撞处理方案也有所不同。

8.5.1 椭圆

椭圆与圆形类似于矩形与正方形之间的关系，也就是说，椭圆可视为圆形在某一方向上的拉伸结果。关于椭圆，一种绘制方法是将绳索固定至纸面的 A、B 两点处，如图 8.10 所示，笔尖位于绳索内部，当围绕两点绘制时即可生成椭圆。椭圆的独特性在于，针对周长上的任意一点 P，AP 和 BP 之和为常量——不难发现，在椭圆绘制过程中，绳长保持不变。

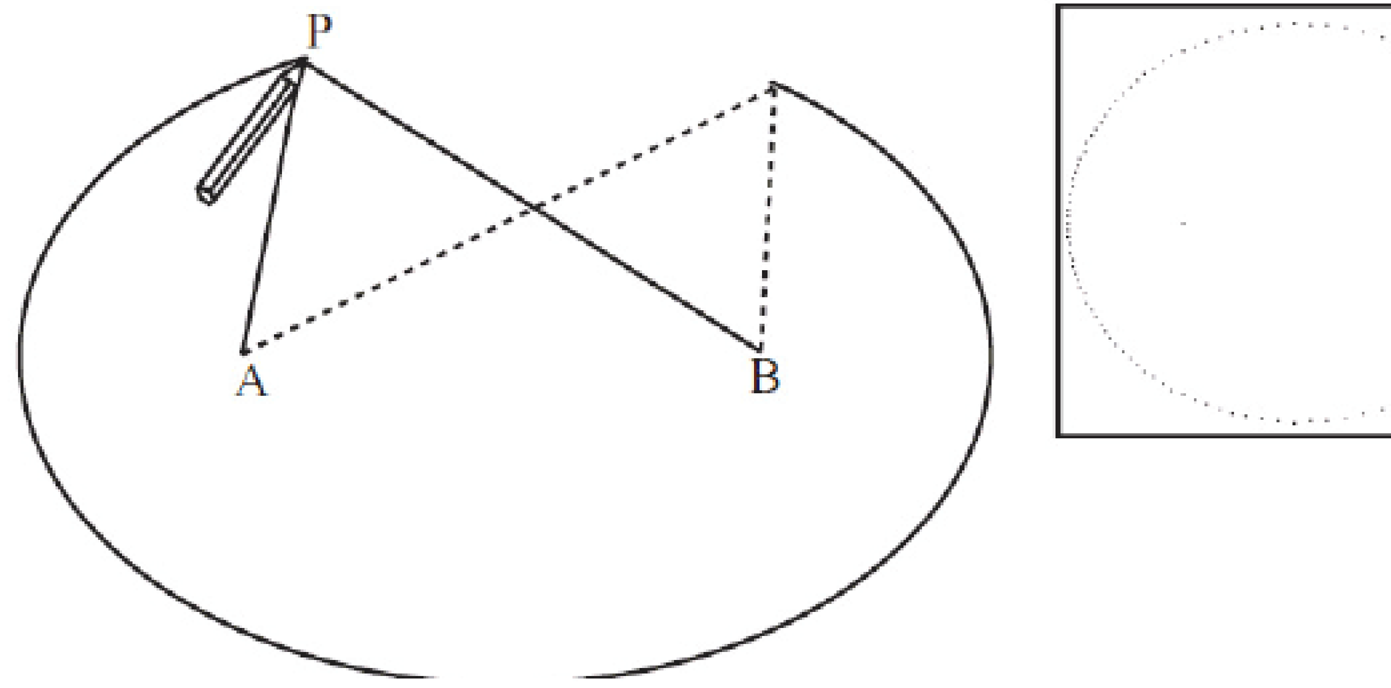


图 8.10　椭圆的绘制过程（当前椭圆示图由计算机生成）

【提示】几何形状的周长即为其边长。例如，圆周表示圆周长。另外，英文单词 foci 为 focus 的复数。

其中，点 A 和 B 称作椭圆的焦点，AP + BP 定义为椭圆的内径。需要注意的是，若 A 和 B 为同一点，则椭圆退化为圆，且内径等于圆的直径。这里，"内径"并非是椭圆的正式称谓，但通过两条直线的和这一概念可明显地区分椭圆和圆形之间的差异。

相应地，存在多种方式可编写椭圆的绘制函数，drawEllipseByFoci()函数即是其中之一，该函数根据既定焦点和内径生成椭圆图案。需要说明的是，当前实现方案使用了第 5 章讨论的 angleBetween()函数，椭圆的绘制过程如下所示：

```
function drawEllipseByFoci(focus1, focus2, diameter)
  set resolution to 100
  //increase this number to draw a more detailed ellipse
  set angle to 2*pi/resolution
  set angleOfAxis to angleBetween(focus2-focus1, array(1,0)
  if angleOfAxis="error" then set angleOfAxis to 0
  set d to magnitude(focus1-focus2)
  set tp to diameter*diameter-d*d
  repeat for i=1 to resolution
  set a to angle*i //the angle made at focus1 with the major axis
  set k to tp/(2*(diameter-d*cos(a)))
  set ha to a+angleOfAxis
  set p to k*array(cos(ha),sin(ha))
  draw point p
  end repeat
end function
```

drawEllipseByFoci()函数并未采用纯数学方法绘制椭圆，读者可自行尝试。对此，关键之处在于根据三角形 PAB 使用余弦定理。纯数学方案将围绕某一焦点绘制更多的数据点，这也可视为该方案的一个缺点。图 8.10 的右图即显示了基于数学方案的绘制结果，与椭圆右侧相比，左侧点则更为密集。

8.5.2　通过坐标描述椭圆

与 8.5.1 节中的方案不同，这里通过坐标值对椭圆加以描述。回忆一下，针对圆心位于(0,0)

且半径为 r 的圆，其上任意一点的坐标表示为$(r\cos\theta, r\sin\theta)$，类似的描述结果也适用于椭圆。基于坐标$(a\cos\theta, b\sin\theta)$的各点可定义一个椭圆，其内径为$\frac{b^2}{2a}$，且焦点分别位于$\left(\frac{b^2-a^2}{2a},0\right)$和$\left(\frac{a^2-b^2}{2a},0\right)$。假设 $a > b$，则 a 和 b 的长度分别称作长半轴和短半轴。

然而，该方案仅可绘制与 x 轴对齐的椭圆，即椭圆沿其主轴对齐。其中，主轴表示为两个焦点之间的向量。若期望椭圆旋转且不再与主轴对齐，则椭圆的绘制工作将变得相对复杂。对此，各点可围绕椭圆中心旋转固定值。在数据点调整函数的辅助下，drawEllipsesByAxes()函数通过坐标值绘制椭圆，如下所示：

```
function drawEllipseByAxes(center, a, b, alpha)
  set resolution to 100 //increase to draw more accurately
  set ang to 2*pi/resolution
  repeat for i=1 to resolution
  set angle to ang*i
  set p to rotateVector(array(a*cos(angle), b*sin(angle), alpha)
  draw point center+p
  end repeat
end function
```

rotateVector()函数关注椭圆的旋转操作。除了易于理解之外，代码重构为两个函数以使rotateVector()函数可用于其他场合中。该函数如下所示：

```
function rotateVector(v, alpha)
  set x to v[1]
  set y to v[2]
  set l to sqrt(x*x + y*y)
  set x1 to l*cos(alpha-atan(y,x))
  set y1 to l*sin(alpha-atan(y,x))
  return array(x1,y1)
end function
```

8.5.3 平移操作

在 8.5.1 和 8.5.2 节中，两种椭圆绘制方案十分相似，但在不同场合中，对应方案均包含各自的优点。相比之下，第二种方案则更为有效，该方案体现了一种理念，即椭圆可表示为拉伸后的圆形。为了准确地描述这一概念，可根据单位圆创建一个位于原点处的椭圆。对此，可在某一方向上通过因子 a 进行缩放，并在另一方向上采用因子 b 执行缩放操作。随后，可旋转椭圆，进而将其置于正确方向上。该过程可通过标准的转换矩阵 **T** 加以描述，矩阵 **T** 涵盖了缩放行为和旋转操作。最后，可将当前椭圆移至对应位置处，并使用 E(**c**,**T**)符号表示此类椭圆。相应地，drawEllipseFromMatrix()封装了前述 drawEllipseByAxes()函数并实现了上述操作，如下所示：

```
function drawEllipseFromMatrix(pos, mat)
  set v to mat.column
```

```
  set n to mat.column
  drawEllipseByAxes(magnitude(v), magnitude(n), atan(v[1],v[2]))
end function
```

椭圆中较为常见的数值即离心率。离心率越大，则椭圆变得越发尖锐。当离心率为 1 时，则椭圆演变为圆形；若离心率趋于无穷大，则椭圆变为一条直线。

椭圆的最后一个特征是，点 P 处的曲线切线与直线 AP 和 BP 呈相同角度。若在椭圆形台球桌面上的焦点处分别搁置一个母球和落袋，则无论从任何角度击球，母球均会落入袋中。在与 x 轴对齐的椭圆上的点$(a\cos\theta, b\sin\theta)$处，其切线位于$(-a\sin\theta\ b\cos\theta)^T$方向上。

8.5.4　静态椭圆和动态点

假设位于点 P 处的粒子以速度 **v** 在椭圆空间内运动，该空间由椭圆 E(**c**,**T**)所占据。这里的问题是，对应点是否会进入椭圆内。为了回答这一问题，可首先考虑平移椭圆的参考坐标系并减去 **c**，最终向量算式如下所示：

$$\mathbf{Tu} = \mathbf{p} - \mathbf{c} + t\mathbf{v}$$

若通过第 5 章讨论的方法逆置矩阵 **T**，则可得到如下方程：

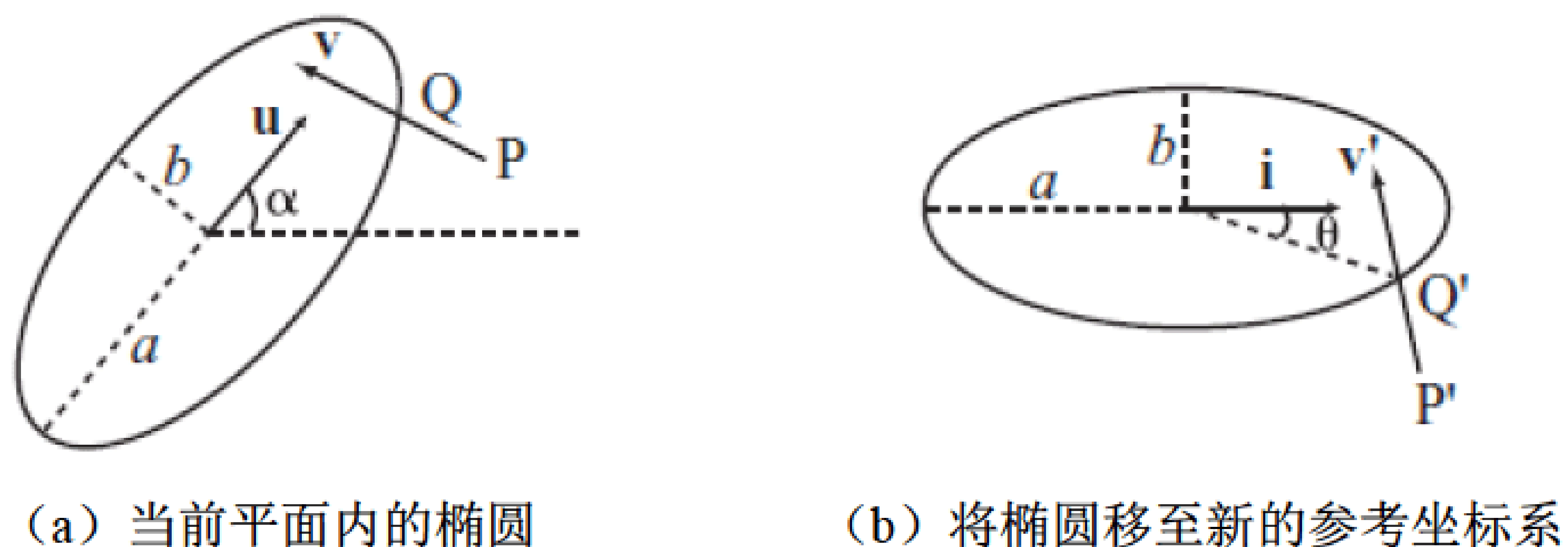
$$\mathbf{u} = \mathbf{T}^{-1}(\mathbf{p} - \mathbf{c} + t\mathbf{v}) = \mathbf{T}^{-1}(\mathbf{p} - \mathbf{c}) + t\mathbf{T}^{-1}(\mathbf{v})$$

由于 **u** 为单位向量，则自身点积为 1，最终方程如下所示：

$$(\mathbf{T}^{-1}(\mathbf{p} - \mathbf{c}) + t\mathbf{T}^{-1}(\mathbf{v})) \cdot (\mathbf{T}^{-1}(\mathbf{p} - \mathbf{c}) + t\mathbf{T}^{-1}(\mathbf{v})) = 1$$

相信读者对上式不会感到陌生，即粒子与圆之间的交点计算，该问题已在前述内容中予以完善处理。

当编写椭圆与粒子之间的碰撞处理函数时，可从几何角度加以考察。图 8.11（a）生成了一个椭圆，而图 8.11（b）则将该椭圆旋转至新的参考坐标系，且各轴与基向量对齐。最后一个步骤则并未显示于图中，读者可压缩空间并将椭圆转换为圆。虽然各项数据按照位移向量运动，但沿直线上的相对距离保持不变（即使绝对距离产生了变化），这也体现了相对性原理的功效。



（a）当前平面内的椭圆　　（b）将椭圆移至新的参考坐标系

图 8.11　静止椭圆和运动点

particleEllipseCollision()函数根据前述讨论实现了一类处理方案，其参数为点数据和椭圆。类似于本章中的其他函数，该函数也使用了第 5 章所讨论的相关函数，具体内容如下所示：

```
function particleEllipseCollision(pt, ell)
  set t to ell.transformationMatrix
```

```
  set inv to inverseMatrix(t)
  set p to pt.pos-ell.pos
  set w to matrixMultiply(inv,p)
  set ww to dotProduct(w,w)
  if ww<1 then return "inside"
  set v to matrixMultiply(inv,pt.displacement-ell.displacement)
  set a to dotProduct(v,v)
  set b to dotProduct(w,v)
  set c to ww-1
  set root to b*b-a*c
  if root<0 then return "none"
  set t to (-b-sqrt(root))/a
  if t>1 or t<0 then return "none"
  return t
end function
```

8.5.5　两个椭圆之间的碰撞

如果在同一方向对齐的两个椭圆正面碰撞，其碰撞行为与前述圆形示例十分类似。在更为通用的场合中，若两个椭圆 E($\mathbf{p}$,$\mathbf{S}$)和 F($\mathbf{q}$,$\mathbf{T}$)以任意速度碰撞，则情况将变得较为复杂。如前所述，可通过相对性原理简化计算任务，即静态圆与椭圆 F'($\mathbf{q} - \mathbf{p}$, $\mathbf{S}^{-1}$ $\mathbf{T}$)碰撞。然而，在当前示例中，该原理并未起到预期的作用。最终结果为较为复杂的非线性联立方程组，且难于求解。

对此，一类相关技巧可描述为，在碰撞点处，两个表面的法线处于平行状态——这可视为全部问题的关键部分，具体细节问题则留于第 19 章进行讨论。

8.5.6　碰撞点

这里，可采用与圆形类似的方案处理椭圆的碰撞点，但依然需要谨慎处理某些问题。根据各函数返回的 t 值，可方便地计算移动对象的碰撞位置。然而，若精确地确定几何形状上的碰撞位置，则需要将该对象转换回正确的参考坐标系中，因而应在转换后的坐标系中计算碰撞点，并于随后将该点通过 $\mathbf{T}$ 再次进行转换。

8.6　不同形状对象间的碰撞

最后一节将讨论不同形状之间的碰撞行为，并主要考察圆形和矩形之间的碰撞结果。

8.6.1　圆形和矩形之间的碰撞

针对同一类型的对象，其计算过程相对简洁；然而，对于更为通用的场合，情况又当如何？

相关结论表明，相似形状的计算过程同样适用于混合形状。在图 8.12 中，圆 C(**p**,*r*)以速度 **v** 与矩形 R(**u**, **a**, **b**)正面碰撞。需要注意的是，随着 **v** 不断变化，具体情况分为下列 3 种情形：

- C 与 R 的顶点碰撞。
- C 与 R 的边碰撞。
- 二者未产生碰撞。

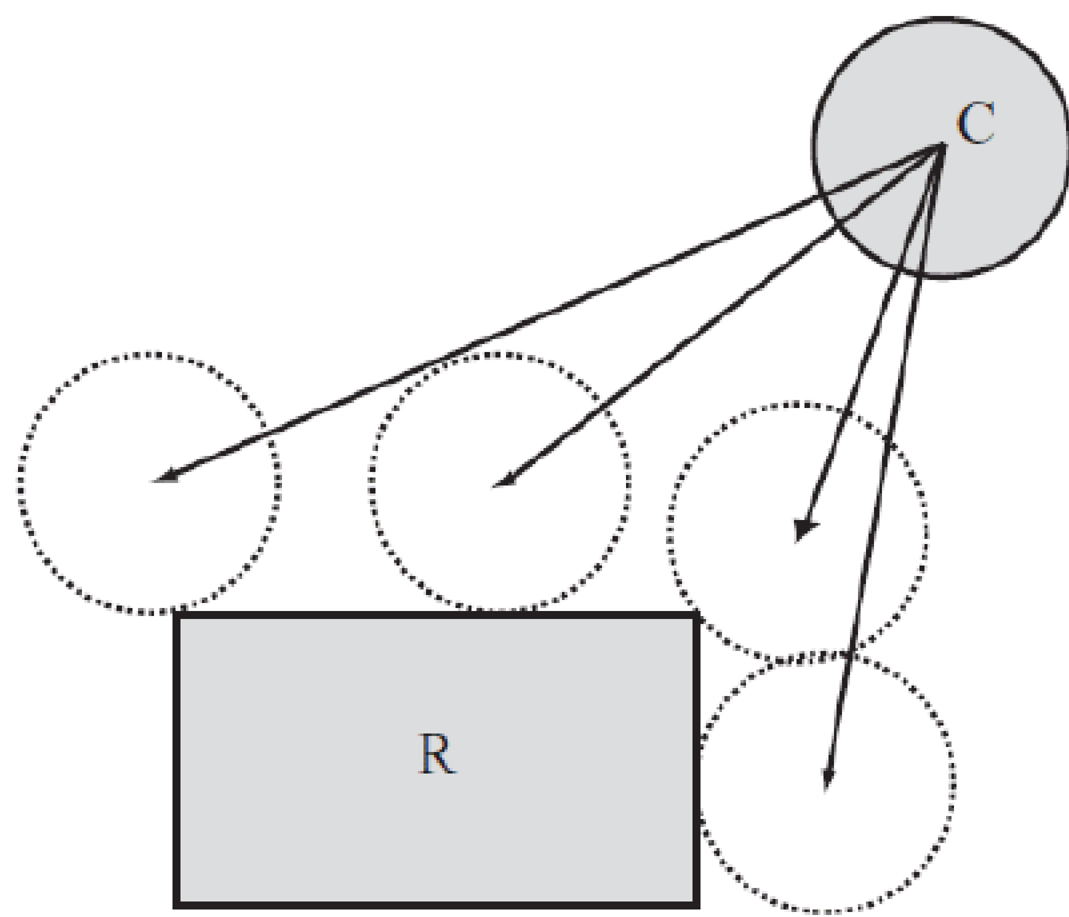


图 8.12　圆形和矩形之间的碰撞

针对第 1 项内容，读者可考察 R 的顶点，并通过 pointCircleCollision()函数检测与圆 C 之间的交点，进而计算碰撞点。对于第 2 项内容，可将 R 中的边视为直线段或一个墙面，并采用 circleWallCollision()函数对其进行处理。除此之外，还可通过圆半径在各方向上对矩形进行扩展，并于随后使用 pointRectangleCollision()函数。在上述两个示例中，预先计算潜在碰撞的顶点或边，这一点与矩形-矩形碰撞处理过程较为相似。而对于第 3 项内容，目前尚无对应函数以及相关需求。

8.6.2　碰撞点

关于碰撞点，圆和矩形之间的碰撞处理与矩形-矩形基本相同。无论碰撞结果为边或顶点，法线通常为碰撞点处的圆法线。然而，当碰撞过程涉及边时，边法线计算往往相对简单，

8.7　本 章 练 习

【练习 8.1】 试编写 pointParallelogramCollision()函数，并采用 pt，displacement，parrPos，side1，side2 作为参数，以及 pointRectangleCollision()函数作为模块。pointParallelogramCollision()函数须返回 0～1 之间的值或“no intersection”字符串。

针对该函数，读者应了解以下事实：平行四边形通过平面斜转换可生成矩形。读者可在函数中实现这一功能，或调用 pointRectangleCollision()函数。

【练习 8.2】 试编写包含对应参数的 rectangleRectangleInnerCollision()函数、circleRectangle

InnerCollision()函数以及 rectangleCircleInnerCollision()函数，并对几何形状内部的碰撞予以检测。

类似于圆-圆碰撞示例，内部碰撞与外部碰撞检测十分类似，但复杂程度视具体情况而定。读者可参考本章代码以获取相关信息。

8.8 本章小结

本章提供了大量的与程序设计有关的建议，尽管如此，碰撞检测依然处于开始阶段。第 9 章将讨论两对象碰撞后的处理过程，随后，读者可针对复杂形状的碰撞检测采取更为通用的计算方案。

至此，读者应掌握如下内容：

- 如何以碰撞检测形式描述圆形、矩形、椭圆以及平面直线。
- 不同形状组合间的碰撞检测。
- 粒子的碰撞检测。
- 墙面的碰撞检测。
- 同类对象间的碰撞检测。
- 椭圆通常难于处理，因而在处理通用场合椭圆碰撞计算之前，须对其进行深入分析。

第 9 章　碰撞处理方案

本章包含如下内容：

- 概述。
- 处理单一碰撞行为。
- 处理多次碰撞行为。

9.1　概　　述

在考察不规则形状的复杂碰撞行为之前，有必要讨论广泛意义上的碰撞行为，此类行为并不涉及深入的物理和数学知识。在本章中，该问题体现在如何确定两个对象碰撞后的运动行为，即碰撞处理方案。

第 8 章曾讨论了与碰撞相关的大多数内容。其中，复杂碰撞涵盖了旋转操作，而简单的碰撞操作无须过多关注碰撞对象的形状。相反，质量、碰撞对象的速度以及碰撞法线往往视为问题的核心内容。本书第 26 章将探讨如何计算碰撞点以及碰撞点处的法线或切线，而本章将相关内容直接应用于实际操作中。

9.2　处理单一碰撞行为

碰撞的基本原理包括能量守恒和动量守恒，其中最为简单的一类碰撞是弹性碰撞。弹性碰撞较为抽象，且较少出现于真实世界中。弹性碰撞仅阐述了理论情形，在该场合下，碰撞前后的动能保持守恒。

实际上，碰撞后的动能通常会发生变化，皆因碰撞过程中对象会释放能量。例如，在大气环境下，碰撞对象中的分子运动加剧，该能量以热能方式释放，并对周围空气产生影响。通常情况下，部分能量还可以声音方式被人们所感知。即使在真空环境中，对象间的碰撞也并非是完全弹性碰撞，期间总会有能量被释放。

尽管真实事件往往包含诸多限制条件，但出于简单性考量，碰撞处理的初始阶段往往会使用到弹性碰撞，并在此基础上进一步处理真实的碰撞行为。

9.2.1　球体与墙面之间的碰撞

图 9.1 显示了以速度 **v** 运行的球体与静止墙面（法线为 **n**）之间的碰撞过程，作为弹性碰撞

的一个简单示例，此处无须考察球体和墙面的细节问题。

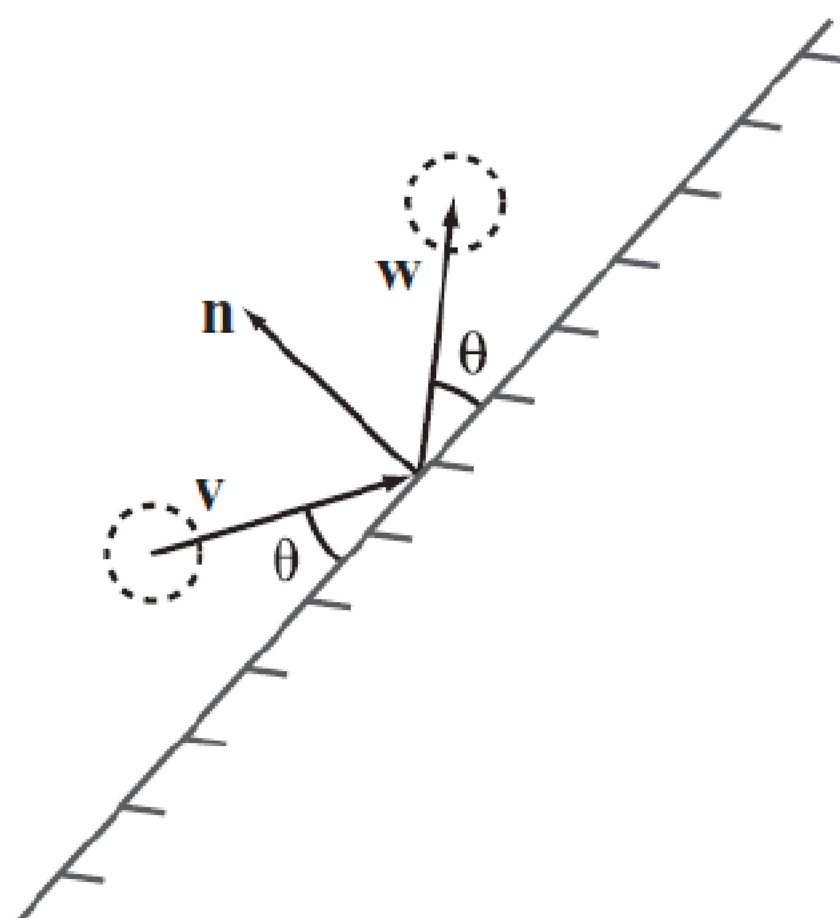


图 9.1　球体与墙面之间的碰撞

针对图 9.1 中的碰撞行为，一种较为简单的方法是将向量 **v** 分解为两个分量，方向 1 为 **n** 方向（法向分量），方向 2 为墙面方向（切向方向）。由于球体在切向方向未受到外力作用，因而球体在该方向上的速度保持不变；而在另一个方向，球体速度则产生逆转。此处，球体能量保持守恒。对此，可简单地减去法线分量两次，进而计算最新速度。对应 resolveFixedCollision()函数如下所示：

```
function resolveFixedCollision(obj, n)
  set c to componentVector(obj.velocity, n)
  set obj.velocity to v-2*c
end function
```

resolveFixedCollision()函数较为简单，并适用于某一对象固定这一类碰撞操作。另外，该函数使用了第 5 章介绍的 componentVector()函数，并可获得碰撞点处的法线数据。根据上述信息，碰撞对象通常会保持同一路径。同时，关于函数的特定计算，读者还可尝试某些代数运算。若 **c** 表示为某一方向上的 **v** 分量，则垂直方向上的分量可表示为 **v−c**。

【提示】通过观察可知，resolveFixedCollision()函数并未返回数据值，其原因在于，对象的 obj 假设对象已被传递至当前函数中，且该对象包含 velocity 属性。通过改变 velocity 中的数据，该函数可隐式地返回数据值。

如图 9.1 所示，resolveFixedCollision()函数表明，原速度向量与墙面之间的夹角（入射角）等同于最终速度向量与墙面之间的夹角（反射角）。

在前述弹性碰撞的讨论中，resolveFixedCollision()函数主要关注碰撞前后球体的运动行为，该函数并未涉及碰撞过程中的具体内容，该问题需要处理大量的细节内容，第 12 章将对此予以讨论。

9.2.2　球体与运动的墙面发生碰撞

当对象处于运动状态时，情况则变得越加复杂。若两个对象均可移动，则需要确定两个对象

碰撞后的速度。当深入分析此类问题时，则需要使用到更多的信息。

滚珠-炮弹与落锤-炮弹之间的碰撞计算并无太大差异，为了制定正确的求解方案，此处需要了解两个对象的质量。更为重要的是，能量和动量信息同样不可或缺。若两个对象皆处于运动状态，首先，碰撞前后的动量保持不变，其次，碰撞前后的全部能量亦保持守恒。

对此，一类较为简单的示例可描述为：某一对象碰撞前的速度为 0——稍后将会看到，根据相对性原理，在执行速度减法运算后，其他运动状态也可转化为这一情形。这里，假设球体 B（入射球体）的质量为 m 且速度为 $\mathbf{u}$，并撞击质量为 p 的静态可移动球体 C，对应碰撞法线表示为 $\mathbf{n}$。待碰撞后，如图 9.2 所示，球体 B 的速度为 $\mathbf{v}$，而球体 C 的速度为 $\mathbf{w}$，且二者均为未知项。

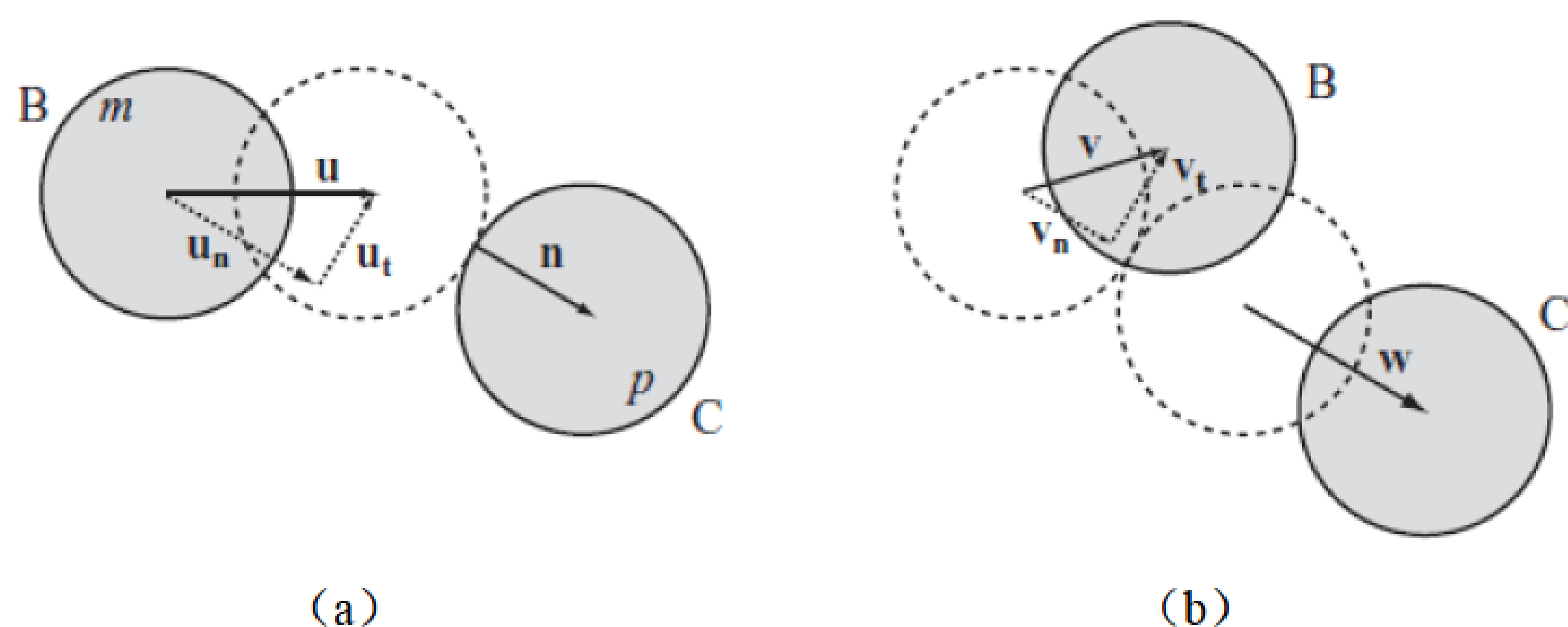


图 9.2　与静止的可移动对象之间的碰撞

如前所述，可将 $\mathbf{u}$ 划分为两个分量，其大小分别为 u_n（法向）和 u_t（切向），该操作同样适用于 $\mathbf{v}$ 和 $\mathbf{w}$。由于切向并不存在作用力，因而球体在切线方向上的速度保持不变。最终，v_t=u_t，w_t=0。在其他方向，通过动量守恒可知：

$$mu_n=mv_n+pw_n$$

为了确保后续计算简单，此处可除以 p 并得到比率值 $r=\dfrac{m}{p}$。据此，可得到 $ru_n = rv_n + w_n$。

同时，通过能量守恒定律可知 $\frac{1}{2}mu^2 = \frac{1}{2}mv^2 + \frac{1}{2}pw^2$，经分解后可得到如下算式：

$$r(u_n^2+u_t^2)=r(v_n^2+v_t^2)+(w_n^2+w_t^2)$$

针对切线法向，存在如下算式：

$$r(u_n^2+u_t^2)=r(v_n^2+v_t^2)+w_n^2$$

当前，可得到基于未知项 v_n 和 w_n 的联立方程，并可通过替代法进行求解。根据动量方程可得到下列算式：

$$w_n=r(u_n-v_n)$$

相应地，将该值代入能量方程中，则可得到如下算式：

$$r(u_n^2+u_t^2)=r(v_n^2+u_t^2)+r^2(u_n-v_n)^2$$

$$u_n^2+u_t^2=v_n^2+u_t^2+ru_n^2-2ru_nv_n+rv_n^2$$

$$(r+1)v_n^2+(r-1)u_n^2-2ru_nv_n=0$$

至此，可得到基于 v_n 的二次方程，经分解后可得到下列算式：

$$(v_n-u_n)((r+1)v_n+(r-1)u_n)=0$$

上述方程包含两个根值，即 $v_n = u_n$ 和 $v_n = \frac{r-1}{r+1}u_n$。其中，第一个根为期望值，该值反映了初始碰撞环境。另一个根为碰撞后的计算结果。需要注意的是，这取决于球体质量比率，而非其绝对值。同时，u_t 值从当前计算中消失，这也是期望中的计算结果，此处，两个分量间应彼此无关。

【提示】弹性碰撞具有时间可逆性这一特征，因而初始环境应提供有效的方程解。换而言之，若逆置碰撞后的全部速度值，并于随后再次运行当前碰撞环境，相关对象应位于其初始位置。

当前，可将相关值代入动量方程中，进而求解 w_n，如下所示：

$$w_n = r\left(u_n - \frac{r-1}{r+1}u_n\right)$$
$$= \frac{2ru}{2+1}$$

resolveCollisionFree1()函数负责处理运动对象。类似于前述函数，该函数使用了第 5 章讨论的 componentVector()函数。这里，两个对象被传入至函数中，且各对象的速度均已发生改变。对应函数如下所示：

```
function resolveCollisionFree1(obj1, obj2, n)
  set r to obj1.mass/obj2.mass
  set un to componentVector(obj1.velocity,n)
  set ut to obj1.velocity-un
  set vn to un*(r-1)/(r+1)
  set wn to un*2*r/(r+1)
  set obj1.velocity to ut+vn
  set obj2.velocity to wn
end function
```

9.2.3　两个运动球体的碰撞

前述示例仅限于使用单一对象的速度数据。向第二个对象添加速度数据较为直接，可在计算前从全部对象中减去速度值。实际上，此类算法在前述内容中已有所讨论。待减去该速度后，还应在后续计算中再次添加。resolveCollisionFree()函数即采用了这一方案，如下所示：

```
function resolveCollisionFree(obj1, obj2, n)
  set r to obj1.mass/obj2.mass
  set u to obj1.velocity-obj2.velocity
  set un to componentVector(u,n)
  set ut to u-un
  set vn to un*(r-1)/(r+1)
  set wn to un*2*r/(r+1)
  set obj1.velocity to ut+vn+u2
```

```
  set obj2.velocity to wn+u2
end function
```

在 resolveCollisionFree()函数中，除了碰撞法线计算，对象的形状和尺寸并未受到任何影响。该函数可处理各种弹性碰撞行为。

进一步讲，若 r=1，则两对象的质量相等，则有 v_n=0 且 w_n=u_n。这体现了一种“牛顿摆（Newton’s Cradle）”效应，其中，某一球体的速度完全“传递”至另一个球体，而原球体保持静止。对此，resolveCollisionEqualMass()函数实现了这一功能，如下所示：

```
function resolveCollisionEqualMass (obj1, obj2, n)
  set u to obj1.velocity-obj2.velocity
  set un to componentVector(u,n)
  set ut to u-un
  set obj1.velocity to ut+obj2.velocity
  set obj2.velocity to un+obj2.velocity
end function
```

针对 resolveCollisionEqualMass()函数，若 r 趋于 0，则 $v_n \to u_n$ 且 $w_n \to 0$。也就是说，质量 p 相对于 m 趋于无穷大，该结果类似于前述固定墙面示例。当球体质量趋于无穷大时，则该对象处于固定状态。换而言之，当前计算演变为固定墙面碰撞。相反，若对象处于固定状态，则该对象的质量可视为无穷大。

9.2.4　非弹性碰撞

在非弹性碰撞中，碰撞前后的能量并不相等。例如，部分损失将转换为热量或声音。

一种最为简单的能量损失模拟方法是设置固定比例的能量，并以此表示两个对象碰撞时的能量损失。这里，该比率值称作效率，并可分别应用于两个对象上。若模拟环境中的对象形状相似，则可使用全局效率；否则，可针对各对象设定效率值，并在每次碰撞行为中对其进行整合。

例如，假设球体 B 在各次碰撞中的能量转换效率为 95%，球体 C 为 90%。若二者的能量值分别为 E_b 和 E_c 且发生碰撞，则碰撞后的全部能量为 $0.95\times0.9\times(E_b+E_c)$。

类似于弹性碰撞，非弹性碰撞也采用了一类简化方案，真实的碰撞和行为并非呈线性状态。总体而言，与慢速碰撞相比，快速碰撞更为高效。需要说明的是，效率会受到诸多因素的影响，例如空气的温度。尽管如此，由于误差处于可接受范围内，因而上述方案依然有效，当对象处于正常的速度和质量范围时尤其如此。

非弹性碰撞的计算方程稍显复杂，但碰撞切向运动并未受到影响，且在法线方向上，动量守恒依然成立。实际上，若未受到外部作用力，该定律总保持有效。另外一方面，应重新审视能量守恒问题，且需要将效率引入至当前计算中，最终算式如下所示：

$$\frac{1}{2}emu^2=\frac{1}{2}mv^2+\frac{1}{2}pw^2$$

其中，e 表示为两个对象效率的乘积，并以分数形式予以呈现。鉴于切向运动未受影响，因而可得到下列算式：

$$er(u_n^2+u_t^2)=r(v_n^2+u_t^2)+w_n^2$$

其中，$r=\dfrac{m}{p}$。替换 w 后则可得到如下算式：

$$(r+1)v_n^2+2ru_nv_n+(r-e)u_n^2+(1-e)u_t^2$$

需要注意的是，除非 $e=1$，否则 u_t 总出现在上式中。若 $e=1$（即全效率），则上式演变为弹性碰撞。切向速度将会对最终结果产生影响，其原因在于，系统包含越多能量，则能量的损失也就越大。

上式包含两个根，并可通过下列二次方程进行计算：

$$v_n=\frac{-ru_n\pm\sqrt{r^2u_n^2-(r+1)((r-e)u_n^2+(1-e)u_t^2)}}{r+1}$$

u_n 不再是当前方程的根值（除非 $e=1$），皆因非弹性碰撞不包含时间可逆这一特征。然而，通过对弹性碰撞的分析可知，特殊根值一般源自负平方根（并替换 w）。resolveInelasticCollisionFree()函数即体现了这种情况，如下所示：

```
function resolveInelasticCollisionFree(obj1, obj2, n)
  set r to obj1.mass/obj2.mass
  set u to obj1.velocity-obj2.velocity
  set e to obj1.efficiency*obj2.efficiency
  set un to component (u,n)
  set ut to mag(u-un*n)
  set sq to r*r*un*un-(r+1)*((r-e)*un*un+(1-e)*ut*ut))
  set vn to n*(sqrt(sq)-r*un)/(r+1)
  set wn to r*(n*un-vn)
  set obj1.velocity to ut+vn+ obj2.velocity
  set obj2.velocity to wn+ obj2.velocity
end function
```

当 e=1 时，resolveInelasticCollisionFree()函数表示为弹性碰撞。同样，在当前示例中有 sq=un*un，这将生成与前述内容相同的结果值。

出于完整性考量，resolveInelasticCollisionFixed()函数定义了固定非弹性碰撞，并在原有函数的基础上将 r 设置为 0。若将 resolveInelasticCollisionFixed()函数与弹性版本进行比较，则会发现二者具有许多相似之处。resolveInelasticCollisionFixed()函数的具体内容如下所示：

```
function resolveInelasticCollisionFixed (obj1, obj2, n)
  set e to obj1.efficiency*obj2.efficiency
  set un to component (obj1.velocity,n)
  set ut to mag(obj1.velocity -un*n)
  set sq to (e*un*un+(e-1)*ut*ut))
  set vn to n*sqrt(sq)
  set obj1.velocity to ut+vn
end function
```

同样，resolveInelasticCollisionFixed()函数与各对象的质量无关，因而易于实现。

9.3　处理多次碰撞行为

严格地讲，本节内容隶属于碰撞检测范畴，而非碰撞处理解决方案。实际上，对象的多次碰撞与二者皆有关系。这里的问题是，如何处理多个运动对象？若多个对象在同一时刻发生碰撞，结果又当如何？

9.3.1　递归碰撞

复杂碰撞检测的关键之处在于理解相应的处理过程，在计算机模拟环境下，该过程可分为 6 个步骤，如下所示：

（1）首先需要确定对象集 O1，O2，…，各对象以某一速度处于运动状态，并包含质量、效率以及其他特征信息。

（2）当前模拟环境可划分为时间步，并通过对应变量加以定义，该值通常取决于计算机的处理速度。同时，通过与最近已知时间步比较，可计算各时间步的大小。当前，时间步表示为 s 个时间单位。

（3）在各时间步内，需要遍历全部对象并获取首个碰撞组合。通常，碰撞检测函数返回位于 0～1 之间的 t 值以及碰撞法线。

（4）截止至碰撞时的时间段表示为 ts，最终，全部对象的移动距离为 ts 乘以对象的速度。

（5）求解碰撞问题。随后，各对象均包含新的速度数据。

（6）在当前时间步内，由于尚存在$(1-t)s$ 个时间单位，因而需要返回至步骤（3）并重复计算，直至不存在其他碰撞。

checkCollision()函数实现了上述步骤，该函数较为冗长，旨在显示各步骤的详细内容。另外，该函数调用的某些函数还取决于读者自身的计算环境，稍后将对此进行讨论。除此之外，还应注意对象列表被划分为固定对象表和可移动对象表，以适当精简当前算法。checkCollision()函数如下所示：

```
function checkCollision(time, movableObjects, fixedObjects)
  //check for the earliest collision
  set mn to 2
  set ob1 to 0
  set ob2 to 0
  repeat for i=the number of movableObjects down to 1
    set obj1 to movableObjects[i]
    //find the displacement vector and other
    //relevant facts about this object
    set l to parameters(obj1, t)
    //search for collisions with other movableObjects
    //(don't bother with those already checked)
    repeat for j=i-1 down to 1
```

```
      set obj2 to movableObjects[j]
      set l2 to parameters(obj2, t)
      set c to detectCollision(l,l2)
      if c is not a collision array then next repeat
      set tm to c[1] //the time of collision
      set m to min(mn,tm)
      if m<mn then
        set mn to m
        set n to c[2] //the normal vector
        set ob1 to obj1
        set ob2 to obj2
        set lf1 to l
        set lf2 to l2
      end if
    end repeat
    //now search for collisions with fixed objects
    repeat for j=the number of fixedObjects down to 1
      set obj2 to fixedObjects[j]
      set c to detectCollision(l, l2)
      if c is not a collision array then next repeat
      set tm to c[1] //the time of collision
      set m to min(mn,tm)
      if m<mn then
        set mn to m
        set n to c[2] //the normal vector
        set ob1 to obj1
        set ob2 to obj2
        set lf1 to l
        set lf2 to l2
      end if
    end repeat
  end repeat
  if mn=2 then set tmove to 1
  otherwise set tmove to mn*t
  repeat for each obj in movableObjects
    moveObject(obj, tmove)
  end repeat
  //if there is no collision you are finished
  if mn=2 then return
  //otherwise, you can resolve the collision here
  set res to resolveCollision(lf1, lf2)
  setNewVelocity(ob1, res[1])
  setNewVelocity(ob2, res[2])
  //and now recurse for the rest of the time-step
  checkCollision(t*(1-mn), movableObjects, fixedObjects)
end function
```

checkCollision()函数中的某些细节内容尚未完成，且与具体的编程环境有关。在面向对象编程设计中，各个处于运动状态的几何形状往往通过对象形式加以描述，某些对象还可能包含于不同形状对应的子类。在过程式程序设计中，则针对各种形状使用数值数组。无论如何，最终处理

过程依然不变。关于某些被调函数，当前环境下的具体内容尚未讨论，这也意味着，某些与计算环境相关的特定工作还需进一步予以优化，当前目标仅是考察整体处理步骤。

另外，通过剪裁操作，函数的工作效率可得到大幅度的提升。具体而言，剪裁操作将在计算之前事先确定行将碰撞的几何形状。除了剪裁操作之外，还应避免重复计算行为。例如，在时间步计算序列中，若对象 A 和 B 在某一时间步中发生碰撞，则在下一个时间步中二者不会碰撞，除非受到某一变速对象的碰撞。相反，若对象未改变运动路径，则在某一时间步内即将发生碰撞的两个对象，将在下一个时间步内彼此碰撞。通常情况下，若最近一次计算中的某一对象未击中其他对象，则当前全部计算内容均保持有效。后续章节将对此进行深入讨论，进而对相关代码执行优化操作。

9.3.2　同时碰撞

截止到目前为止，前述讨论仅涉及两个对象间的单次碰撞，当更多对象同时碰撞时，情况又当如何？例如某一球体撞击其他两个球体，如图 9.3 所示。

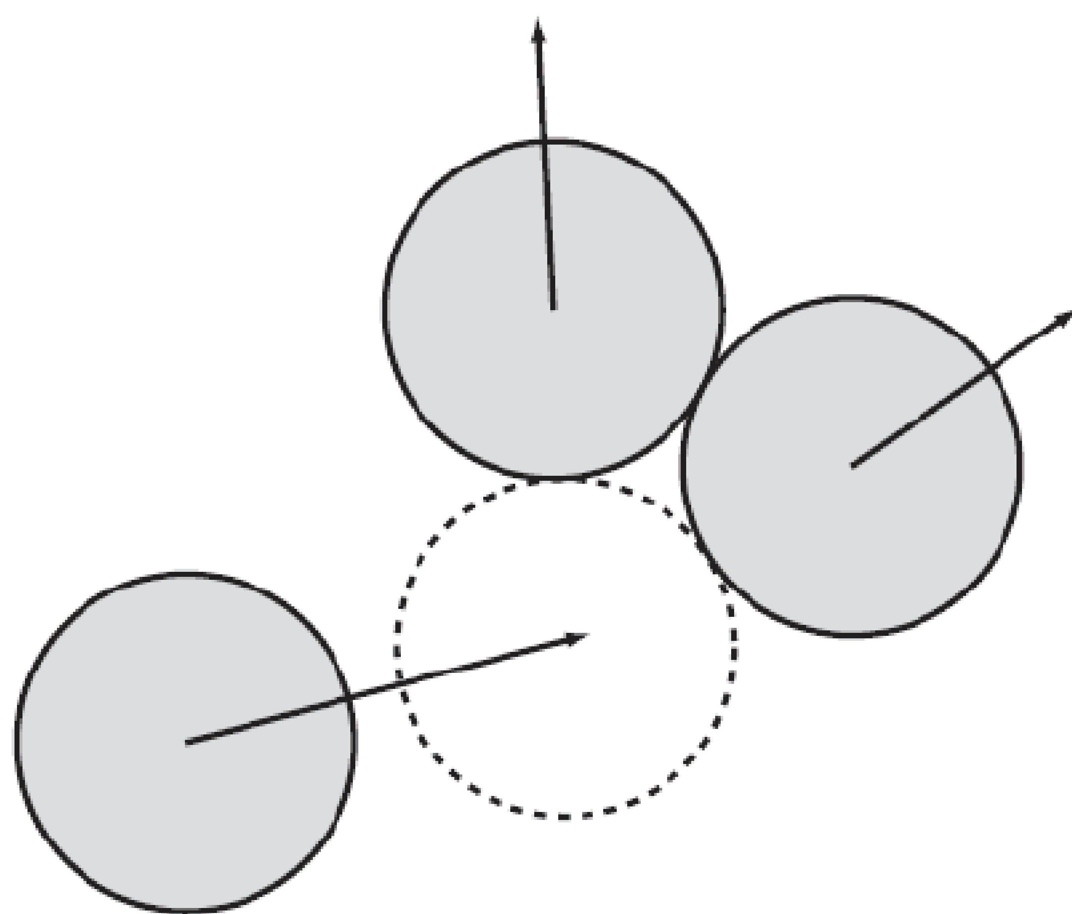


图 9.3　3 个球体同时碰撞

从本质上讲，某一对象于同一时刻与其他两个对象发生碰撞这一情形极少出现，针对全部碰撞行为，单次碰撞检测已然足够。然而，计算机设备的精确度有限，在某一阶段，读者仍需处理同步碰撞问题。

对此，一类较早的处理方式是采用欺骗法，通过扰动方式适当调整模拟参数，以迫使某一碰撞行为于先期出现。实际上，前述函数已对此有所提及，相关函数选取最小碰撞并对其进行处理。

待当前碰撞处理完毕后，第二次碰撞会即刻到来，且有 $t = 0$。这也意味着，碰撞时间不会减少，这将在函数中导致潜在的无限循环问题。然而，多数时候，该问题并不会出现。有时，该问题可能会源自以下情形，即球体沿某一通道滑动（其宽度为球体宽度）。当然，通过调整碰撞环境，可有效地避免此类情况的发生。

如图 9.4 所示，牛顿摆显示了另一种同步碰撞现象。其中，3 个球体并列摆放且彼此接触。此时，若另一球体以速度 **v** 撞击，该入射球体的动量在球体组合之间被传递。最终，最后一个球体以相同的速度运动。这也表明，同步碰撞的相同处理手法亦适用于当前示例。这里，B_1 以速度 **v** 运动，并同时击中球体 B_2（$t = 0$）。同一处理过程呈现为链式传递，直至最后一个球体被碰

撞并以自由方式运动。

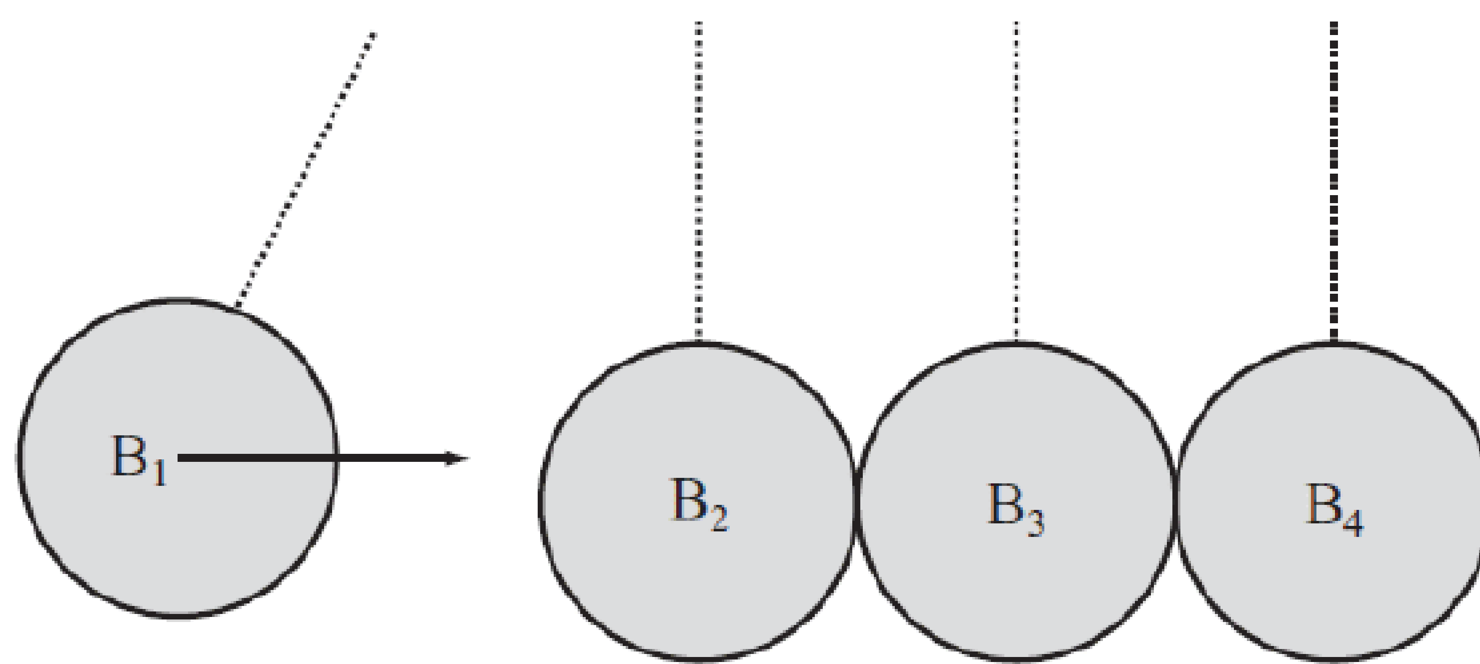


图 9.4　牛顿摆

9.4　本 章 练 习

【练习 9.1】 试完成 checkCollision()函数中的细节内容，特别是函数 detectCollision()和 resolveCollision()函数。

作为编程练习而非数学运算，尝试采用通用方式编写上述函数，生成新的碰撞类型并通过正确的方式对其进行处理。

【练习 9.2】 尝试生成简化版本的牛顿摆模型。对此，可在两端设置两个固定墙面，并于期间放置多组直线排列的球体。同时，可对各组球体设定沿直线的初始速度。最后，通过前述处理函数查看是否可实现牛顿摆效果。

9.5　本 章 小 结

碰撞检测充分体现了数学知识的重要性，另外，数学与物理的有机结合同样值得重视。第 10 章将探讨如何处理复杂形状的碰撞检测。

至此，读者应掌握如下内容：

- 弹性碰撞、非弹性碰撞以及效率的含义。
- 如何采用动量守恒和能量守恒求解两个运动对象（或一个静态对象和另一个运动对象）间的弹性碰撞问题。
- 使用效率系数模拟非弹性碰撞。
- 如何实现简单算法以检测固定对象和可移动对象之间的碰撞行为。
- 如何处理多次同步碰撞行为。

第 10 章　复杂形状间的碰撞检测

本章包含如下内容：

- 概述。
- 复杂形状。
- 某些合理性问题。
- 内建解决方案。

10.1　概　　述

在真实世界中，完美的圆形或矩形之间的碰撞行为较少出现，大多数对象均包含不规则形状。如前所述，即使规则形状的碰撞，例如椭圆，其处理过程也较为复杂。又如，当光滑球体在粗糙地面上弹跳时，同样需要通过多种方式计算碰撞点。

本章将讨论通用碰撞问题的计算方案，包括不规则形状对象。随着过程的不断深入，读者将会发现，依然存在一类简约方案可对此加以处理。期间，读者还将考察地形细节数据的生成和存储方案，包括法线和切向数据的计算。

10.2　复 杂 形 状

复杂形状的碰撞行为为何难以计算？为了回答这一问题，下面首先考察复杂形状的含义，例如，如何正确地描述某些复杂地形？这里，假设球体相对于地面弹跳，进而形成碰撞，抑或球体与角色的脚部发生接触，这在足球游戏中则是一类十分普遍的现象。除此之外，读者还可查看不规则形状之间的碰撞效果，例如足球运动员跌倒于地面之上。综上所述，本章将分析基于此类复杂度的碰撞检测计算。

10.2.1　位图和矢量图

碰撞检测始于计算机与屏幕图像的描述方式，其中包括两种实现方案，且以位图最为简单。位图可通过一定的精确度表示一个数值阵列，并以此定义各像素（图像元素的简称）的位置和颜色值。在大多数场合下，图像复杂度使得图像以逐点方式表示，位图则视为存储图像的唯一有效方式。例如，如何存储 Seurat（或其他画家）绘制的油画作品的照片？需要说明的是，Seurat 发

明了基于大量色点的绘制方式。若制作 Seurat 油画作品的数码照片，则对应数值应赋予各色点中，并以阵列方式加以存储。该阵列确定了各色点的数值和位置。

针对基于阵列方式的独立色点，一种替代方案是使用形状生成算法。该方案可生成较为复杂的图像，然而，若当前目标为重新绘制 Seurat 作品，则该方案的可靠性无法得到保障。尽管如此，针对非复制行为的照片或绘画作品形状，存在多种算法可对其进行处理。其中，一类较为常见的方法是矢量形状。矢量形状通过可选信息加以定义。例如，圆形表示为圆心和半径；矩形则通过 4 个顶点加以标识。

与位图比较，矢量形状具有自身的优、缺点。例如，一个 500×500 的像素正方形需要 250000 个片段信息；相比较而言，矢量形状的正方形则需要大约 10 个片段信息，进而确定 4 个顶点、4 条连接直线以及直线和顶点的颜色值。因此，位图形状所需的信息量约为矢量形状的 25000 倍。除此之外，生成位图形状只需计算机遍历阵列数据，并将对应内容写至显示器即可。相比较而言，矢量形状则需要执行相关计算并重复调用处理器。由于采用计算方式绘制图像，因而矢量形状会降低计算速度。

同样的情况也出现于碰撞检测过程中，也就是说，可通过向量或数据阵列方式存储碰撞信息。关于数据阵列，相应的信息由准确的像素表构成。当出现碰撞时，数据阵列可用作碰撞图（collision map）。作为一般性原理，若基于向量的碰撞描述需要使用到大量的信息，则碰撞复杂度使得向量计算较为低效。通常情况下，若形状越复杂，则逐像素的平面描述方式不失为一种较好的处理方案。某些时候，最佳方案则是对二者进行有效的整合。

10.2.2　定义复杂形状

针对基于位图或矢量的形状定义，下面将以不同形式考察复杂形状。例如，某些形状较为连续但缺乏一定的规则性，而另外一些形状较为规则但缺乏应有的连续特征。多边形即是一例，并可视为一类不规则的连续形状。如图 10.1 所示，多边形由直线段闭环连接的一组顶点构成，即一个随机的凸八边形。这里，凸形意味着其边形成了一个自身闭合的图像，这一点与圆形十分类似。相应地，八边形表示为 8 条边的多边形，也就是说，包含 8 个以既定顺序连接顶点 v_1，v_2，⋯，v_8 的 8 个直线段，其中，位置向量可用于描述直线段。出于简单性考量，此处假设原点为八边形的质心位置（且据此计算位置向量）。

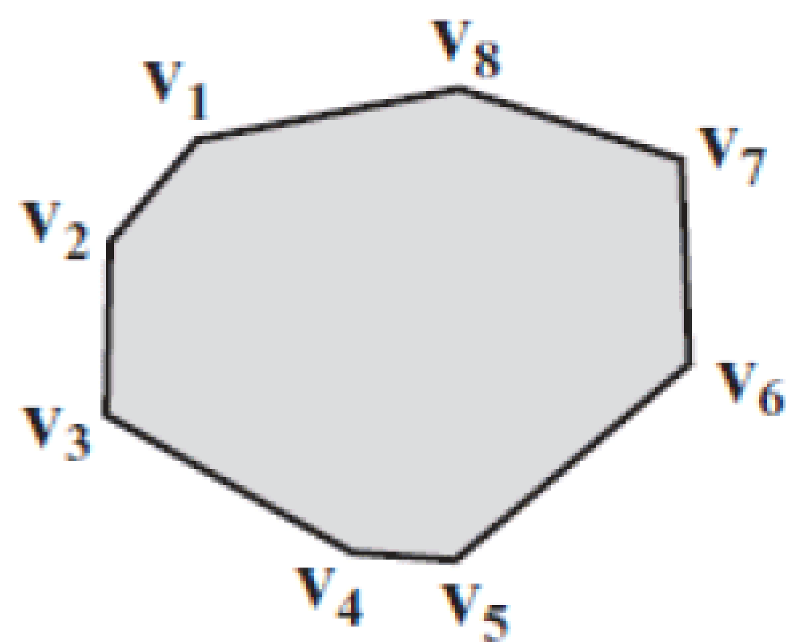


图 10.1　常见的凸多边形

【提示】P(v_1, v_2, ⋯, v_8)表示通用的 n 多边形，即包含 n 条边的多边形。

如前所述，与顶点相比较，另一种复杂形状的描述方式则是使用碰撞图。碰撞图类似于形状位图，黑色像素表示形状内部区域，白色像素表示外部区域。多种因素可影响碰撞图的工作方式，其中一个较为关键的因素则是图像绘制点的密度，即分辨率。图 10.2 显示了虫状生物不同分辨率的碰撞图。尽管碰撞图的计算代价相对高昂，但某些时候却不失为一种最佳方案。通常，读者可通过低分辨率碰撞图节省计算时间，图 10.2 显示了尺寸为 30×30 像素的碰撞图。



图 10.2　碰撞图

10.2.3　碰撞图函数

当使用碰撞图时，可于先期对其进行计算。对此，可采用图像方式加以存储；或者必要时在程序启动阶段予以计算。若在程序初始时进行计算，虽然可有效地降低文件尺寸，但程序的启动时间将变得缓慢。collisionMap()函数展示了碰撞图的生成过程，该函数省略了某些细节内容，此类内容与特定的程序设计语言相关。除此之外，相关方案还与所处理的图像种类相关。collisionMap()函数如下所示：

```
function collisionMap(image, resolution, sensitivity)
  //resolution should be an integer representing
  //the number of pixels of the original per pixel
  //of the collision map.
  //sensitivity should be a float between 0 (no
  //fuzzy edges) and 1 (the whole thing is ignored)
  set map to an empty 2-dimensional array
  set w to ceil((the width of the image)/resolution)
  set h to ceil((the height of the image)/resolution)
  repeat for x=1 to w
    set xstart to (x-1)*resolution
    repeat for y=1 to h
      set ystart to (y-1)*resolution
      set tot to 0
      repeat for i=1 to resolution
        repeat for j=1 to resolution
          add the color of the pixel at (xstart+i, ystart+j) to tot
        end repeat
      end repeat
      divide tot by resolution*resolution
      if tot<(the color of white)*sensitivity then
        set map[x][y] to 1
      otherwise
        set map[x][y] to 0
      end if
    end repeat
  end repeat
  return map
end
```

上述代码还省略了与颜色值-数字转换关系相关的部分代码，其实现方式取决于位图的位深。当前读者可将位深视为 0（黑色）～N（白色）之间的数字。

10.2.4 参数函数

其他方案还包括函数描述法，从某种意义上讲，前述内容曾对该方案有所提及。例如，当描述圆形时，某些信息既已称为已知内容：当且仅当某一点距圆心的距离小于半径，则该点位于圆内。

大多数复杂形状并不包含相对简单的函数描述法，但此类描述法常包含对称特征，图 10.3 显示了一个较为常见的示例，即海星的轮廓线，对应的参数方程如下所示：

$$x = r(\sin(5\alpha) + 2)\cos(\alpha)$$

$$y = r(\sin(5\alpha) + 2)\sin(\alpha)$$



图 10.3 采用简单公式绘制的海星形状

图 10.3 中的海星形状定义为参数方程，其 x 和 y 值皆根据参数 α（以及常量 r）加以描述。此类方程较为常见，实际上，该形状类似于圆形，只是在围绕原点的运动过程中半径值发生变化。

针对上述海星图案，函数描述法的有效性体现在两个方面。首先，此类方法可准确地定义连续形状，并可通过任意分辨率对其进行绘制。这也意味着，读者可在任意碰撞点处确定法线数据。其次，该方法通常易于计算。例如，当确定某一点是否位于边界上时，可将 x，y 代入至方程中并查看计算结果。针对海星形状，可计算 OP 与 x 轴之间的夹角 α 以确定点 P 是否位于内部。对此，须计算 $x = r(\sin(5\alpha) + 2)$并与 OP 长度进行比较。然而，关于碰撞检测，尽管点与形状之间的位置关系易于计算，但这并不会提升碰撞检测的效率。相应地，效率多与计算所涉及的算法相关。

10.2.5 Bezier 曲线和样条

在大多数软件绘制包中，一类常见的参数曲线是 Bezier 曲线。Bezier 曲线表示为曲直线的向量表述方式，并通过两个数据集加以定义，即 v_1，v_2，…，v_n 节点表（或直线上的凸点），以及基于节点的控制点列表，通常记为 c_{12}，c_{21}，c_{22}，c_{31}，c_{32}，c_{41}，…，$c_{(n-1)2}$，c_{n1}。针对开放式曲线，两个控制点应用于一个节点上，且不包括首、尾节点；而闭合曲线节点以非均匀方式与两个节点关联。图 10.4 显示了开放式 Bezier 曲线，并标记了节点和控制点。

在图 10.4 中，控制点和曲线之间的关系较为复杂，并可归结为两种方式。首先，源自节点的控制点方向与曲线相切；其次，控制点的距离体现了曲率，即控制点距离弧线越远，则曲线距

离切线越近。

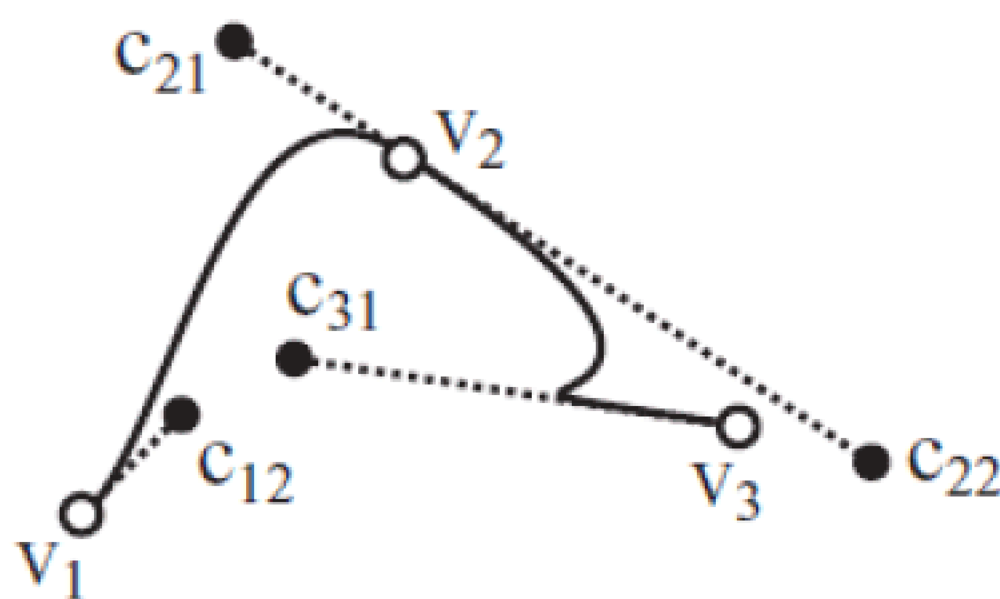


图 10.4　包含 3 个节点的 Bezier 曲线

鉴于较长的曲线可划分为多个部分，各部分内容位于各连续节点对之间，其中，最为简单的曲线考察方式仅涉及两个节点，对应方程为 $n_1 = (x_1, y_1)$，$n_2 = (x_2, y_2)$，$c_{12} = (p_1, q1)$，$c_{21} = (p_2, q_2)$。

类似于海星形状，Bezier 曲线也可通过参数形式加以定义。这也意味着，其形式可根据变量 t 加以描述，该变量位于 0～1 之间。在 $t = 0$ 处，函数计算首个节点；在 $t = 1$ 处，函数计算第二个节点。在两个节点之间，则可得到源自下列 3 次参数方程的连续曲线：

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_1$$

其中

$$c_x = 3(p_1 - x_1)$$
$$b_x = 3(p_2 - p_1) - c_x$$
$$a_x = x_2 - x_1 - 3(p_2 - p_1)$$

同理，类似情形也存在于 $y(t)$中。

若将 0 和 1（t 值）代入至方程中，则生成 x_1 和 x_2。另外，若根据 t 执行微分计算，并再次将 $t = 0$ 代入算式，则可得到 c_2 值。该点处关于 x 的切向值正比于控制点和节点之间的距离。

10.2.6　Catmull-Rom 曲线

Bezier 曲线并非是生成 3 次参数函数的唯一方式，其他方法还包括 Catmull-Rom 样条。针对参数曲线，尽管 Catmull-Rom 样条常作为通用术语加以使用，但该样条并未与控制点协同工作，并采用了一种插值方法（使用位于曲线上的 4 个数据点）。图 10.5 显示了 Catmull-Rom 样条的生成方式，4 个节点 n_0，n_1，n_2，n_3 准确地定义了位于 n_1 和 n_2 之间的曲线段。通过创建包含 $n + 2$ 此类数据点的数据链，则可定义 n 个曲线段。由于各段在端点处包含相同的切线，因而可得到连续的曲线。需要注意的是，由于首、尾控制点无法准确地定义一条线段，因此无法在此类端点处绘制曲线。

类似于 Bezier 曲线，位于 0～1 之间的数据值可通过参数方式定义 Catmull-Rom 曲线，与此对应的样条函数如下所示：

$$p(t)=\frac{1}{2}(2n_1+(-n_0+n_2)t+(2n_0-5n_1+4n_2-n_3)t^2+(-n_0+3n_1-3n_2+n_3)t^3)$$

若代入 $t = 0$ 和 $t = 1$，则可分别得到 n_1 和 n_2。与 Bezier 曲线相比，虽然 Catmull-Rom 样条相对简单，但该样条无法定义独立节点处的尖角形状。

【提示】第 21 章还将介绍一类更为复杂的样条可消除上述限制，即非均匀有理 B 样条（NURBS）。

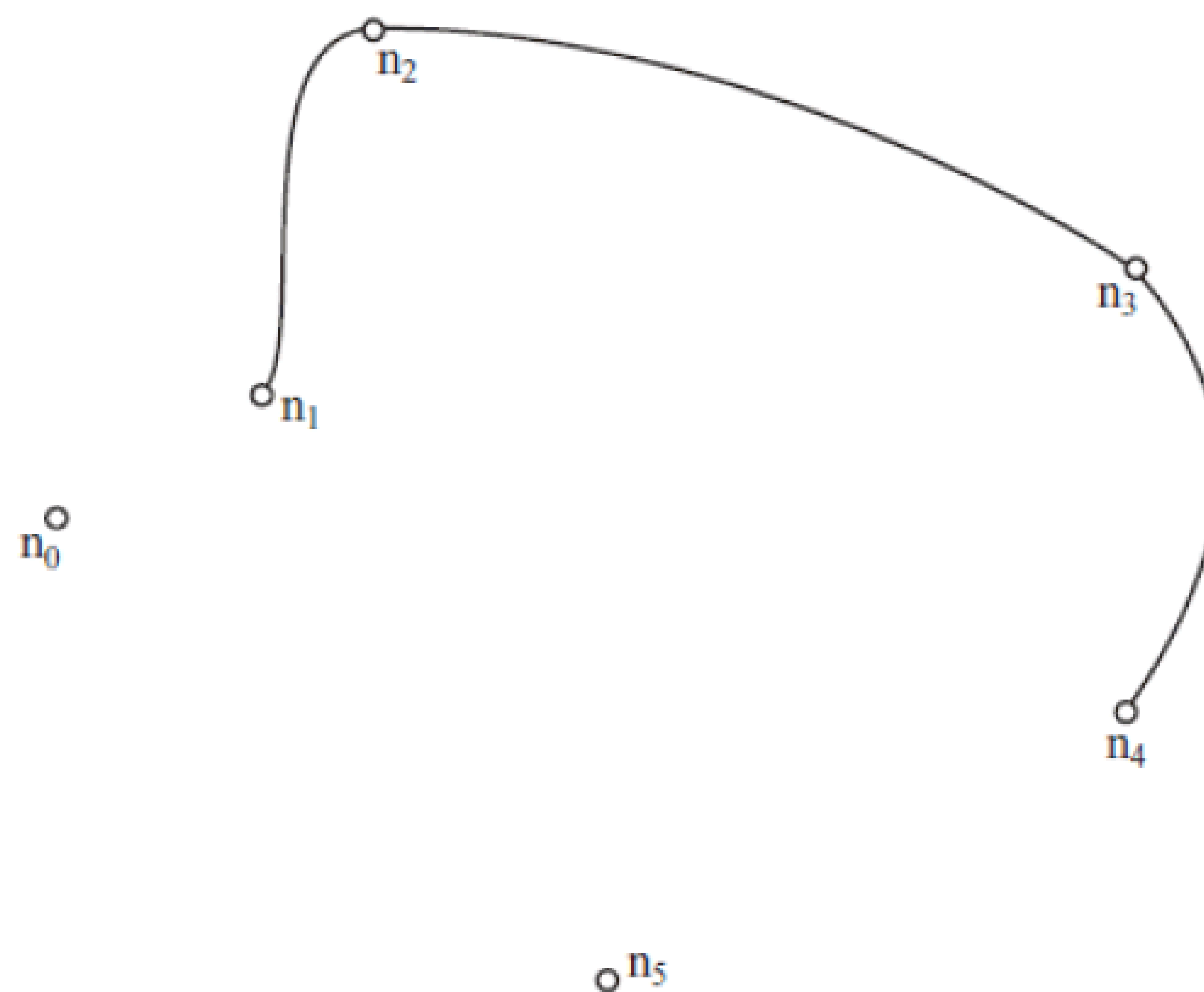


图 10.5　Catmull-Rom 样条

由于 3 次多项式函数相对简单，因而基于样条的碰撞计算易于执行。特别地，还可计算直线与 Bezier 或 Catmull-Rom 样条之间的交点。通常情况下，其目标是计算位于两条直线上的一点。对此，可采用 Bezier 曲线符号并求解下列联立方程：

$$u_1 + sv_1 = a_x t^3 + b_x t^2 + c_x t + x_1$$
$$u_2 + sv_2 = a_y t^3 + b_y t^2 + c_y t + y_1$$

替换 s 后则可得到下列算式：

$$u_1 + \frac{v_1}{v_2}(a_y t^3 + b_y t^2 + c_y t + y_1 - u_2) = a_x t^3 + b_x t^2 + c_x t + x_1$$

$$(v_1 a_1 + v_2 a_x - v_1 a_y)t^3 + (v_2 b_x - v_1 b_y)t^2 + (v_2 c_x - v_1 c_y)t + v_2(x_1 - u_1) - v_1(y_1 - u_2) = 0$$

作为另一个 3 次方程，全部系数均可予以计算。当采用第 3 章介绍的相关方法后，即可求解上述 3 次方程，并以此计算 t 和 s 值。

10.2.7　可移动样条

另一种可解决的问题是处理运动直线和静止样条（或相反）之间的碰撞行为，这需要使用到第 6 章所讨论的参数曲线属性。相关特征可通过基于 t 的微商获得特定 t 值处的曲线斜率，如下所示：

$$\frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{dy}{dx}$$

当计算样条斜率时（进而得到碰撞法线），则可使用下列方案：

$$\frac{dy}{dx} = \frac{3a_y t^2 + 2b_y t + c_y}{3a_x t^2 + 2b_x t + c_x}$$

当计算直线和 3 次参数方程之间的碰撞时，在碰撞处，可假设碰撞点不包括顶点或直线段端点，且直线和曲线沿曲线切线方向相交。最终，曲线斜率等于直线斜率，如下所示：

$$\frac{v_2}{v_1}=\frac{3a_yt^2+2b_yt+c_y}{3a_xt^2+2b_xt+c_x}$$

$$3(v_2a_x-v_1a_y)t^2+2(v_2b_x-v_1b_y)t+(v_2c_x-v_1c_y)=0$$

最终结果为相对简单的二次方程且易于求解，根据运动方向，该方程可在（与直线碰撞的）曲线特定位置处最多生成两个数据点。

为了获取完整的计算结果，可能需要使用到某些额外的计算，具体内容此处不予赘述。例如，数据点是否位于直线段内部；另外，对于包含尖角的 Bezier 曲线，还需进一步计算点-直线与曲线节点之间的碰撞，以及点-样条与直线段端点之间的碰撞。在处理过程中，最终可计算 3 次样条和任意多边形之间的碰撞。

读者可能会认为，若采用合理范围内的（较远）直线，即可计算全部碰撞行为——事实并非如此。对于基于样条的单一对象，或基于多边形的其他对象，借助于近似法，上述函数已然足够。然而，样条与圆形、椭圆形或其他样条之间的碰撞检测则截然不同——从问题的开始阶段，相关计算即处于复杂状态，最终结果可能为 5 阶甚至 6 阶方程，因而难以采用代数方法进行求解。

10.2.8　凸形和凹形

几何形状可划分为两种类型，即凸形和凹形，且存在较为简单的定义方式。如图 10.6 所示，若边界上存在 3 个点 P_1，P_2，P_3，且内角 $P_1P_2P_3$ 大于 180°，则该形状为凹形；相反，若此类点不存在，则对应形状为凸形。上述定义同样适用于多边形和连续形状，另外，上述 3 个点可表示为边界上的任意点。

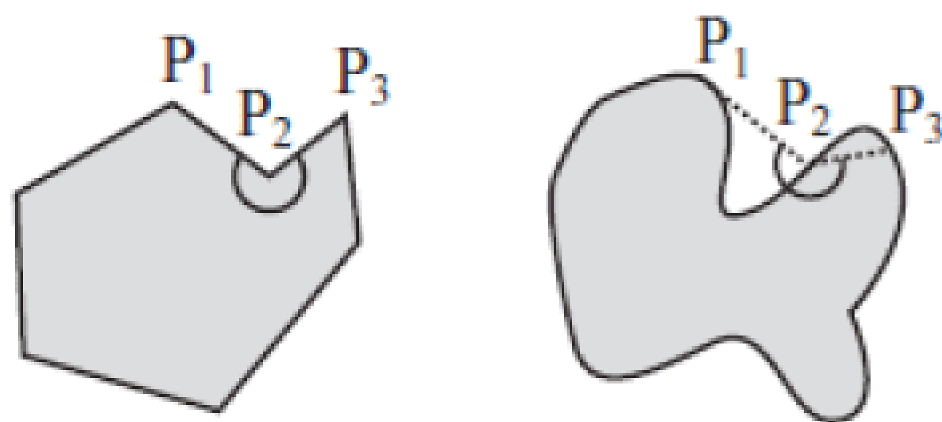


图 10.6　凹形

作为通用规则，凸形通常较为直观。例如，任意直线与图形最多交于两点。对此，读者可通过凸多边形精确定义凸形，随后，碰撞检测与矩形基本相同，仅是检测更多的直线段而已。针对凸形，pointPolygonCollision()函数提供了通用的碰撞检测处理方案，如下所示：

```
function pointPolygonCollision(pt, displacement, poly)
  //here poly is an array of vertex points in order
  set t to 2
  set c to the number of points in poly
  repeat for i=1 to c
  set p1 to poly[i]
  set p2 to poly[(i mod c)+1]
  set t1 to intersectionV(pt,displacement,p1,p2-p1)
```

```
    if t1="none" then next repeat
    set t to min(t,t1)
    end repeat
    if t=2 then return "none"
    return t
  end function
```

经过上述简单讨论可知，pointPolygonCollision()函数与矩形碰撞检测函数十分类似，主要差别仅体现于需要定义更多的边数据。在可能的改进方案中，若可获取当前形状的前缘边（leading edge），则函数的计算速度将得到显著的改观，稍后将对此予以讲解。

pointPolygonCollision()函数适用于凸形对象，而凹形则与此截然不同。广义上的凹形可包含任意复杂度，例如洞穴迷宫。另外，直线可与凹形多次相交，对于某些复杂形状，甚至无法通过多边形对其实施近似处理。实际上，当与分形形状协同工作时，通常难以确定一点是否位于凹形内。需要说明的是，针对本章所讨论的形状，近似多边形方案已然足够。

10.2.9　确定一点是否位于几何形状中

多数时候，该问题易于解决，当几何形状具有函数描述特征时尤其如此；而在其他情况下，则需要使用一类通用方案，例如光线跟踪机制。光线跟踪源自 3D 程序设计，其中，光线可视为某一方向上的无限长直线，这与手电筒发出的光线有几分类似。当采用点/形状碰撞例程时，即可确定光线与目标形状之间的交点。

若点 P 位于几何形状 S 内部，当通过该点向无穷远处投射一条光线，情况又当如何？如图 10.7 所示，由于 S 为封闭直线，因而在某一阶段，图中几何形状与 S 相交。这里，若 S 为凹形，则光线将于某处再次与当前几何形状相交，因此，光线将再次位于几何形状内部——据此可知，相交情形还会于随后再次出现。通过观察可知，若光线始于几何形状内部，则光线与几何形状相交奇数次；相反，若光线从几何形状外部投射，则相交次数为 0 或偶数。

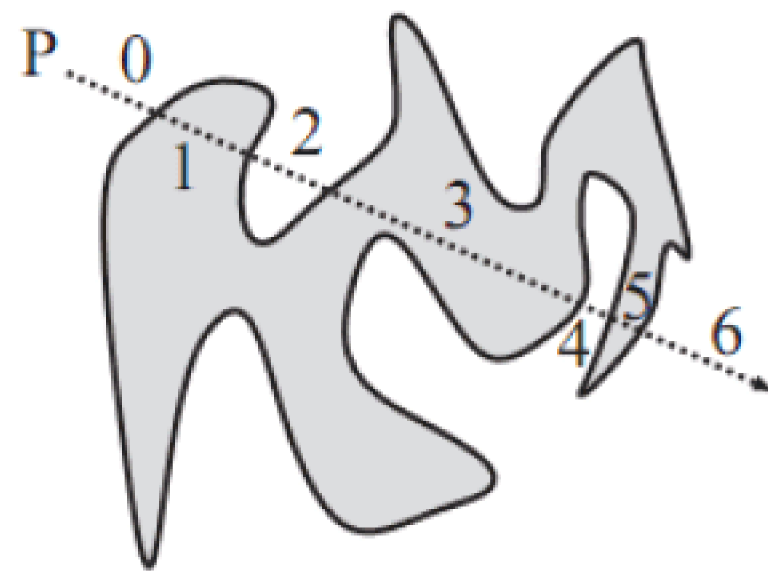


图 10.7　投射一条光线，进而确定某一点是否位于几何形状内部

除此之外，还存在另一种情况，即光线与 S 相切。此时，光线与几何形状接触但并不相交。若 S 为多边形，则可对这一特例进行处理；否则，读者需要投射两条或多条光线。若包含较小夹角的两条光线与点偏差之间处于可接受范围，则对于简单形状而言，其结果可行。否则，还需投射第 3 条光线以获得对应答案。

pointInsidePolygonIncomplete()函数负责测试直线与多边形之间的交点，但并不计算切线这一特殊情况，如下所示：

```
function pointInsidePolygonIncomplete(pt,poly)
  //choose an arbitrary point outside the polygon
  set mx to the maximum x-value in poly
  set outpoint to (mx+10,0)
  //now count the intersections along the ray from pt to outpoint
  set intersections to 0
  set c to the number of points in poly
  repeat for i=1 to c
    set p1 to poly[i]
    set p2 to poly[(i mod c)+1]
    set t to intersection(p1,p2,pt,outpoint)
    if t="none" then next repeat
    add 1 to intersections
  end repeat
  if (intersections mod 2)=1 then return true
  otherwise return false
end function
```

若光线与边界相交但并未穿越几何图形，则情况又当如何？对此，首先可考察光线沿某一边与边界相交这一情形。若光线未经过该边，则必与其平行。针对于平行状态，光线有可能穿越某一顶点。实际上，该光线穿越两个顶点。根据上述信息，全部工作可归结为，光线于何处与顶点相交。

不难发现，pointInsidePolygonIncomplete()函数和早期碰撞检测函数之间存在细微差别。若光线与某一顶点相交，该例程将忽略这一情形且对此不予记录——该结果并不正确，并可在处理切线问题时对其进行修复。在 pointInsidePolygon()函数中，若光线交于某一顶点处，则丢弃该光线并尝试另一条稍显不同的光线。由于当前处理的形状为多边形，因而包含有限数量的顶点。该技术可确保获得正确的计算结果，对应函数如下所示：

```
function pointInsidePolygon(pt, poly, outpoint)
  if outpoint is not defined then
    //choose an arbitrary point outside the polygon
    set mx to the maximum x-value in poly
    set outpoint to (mx+10,0)
  end if
  //now count the intersections along the ray from pt to outpoint
  set intersections to 0
  set c to the number of points in poly
  repeat for i=1 to c
    set p1 to poly[i]
    set p2 to poly[(i mod c)+1]
    set t to intersection(p1,p2,pt,outpoint)
    if t="none" then next repeat
    if t=0 or t=1 then
      //try a different ray
      return pointInsidePolygon(pt, poly, outpoint+(0,100))
    end if
    add 1 to intersections
  end repeat
```

```
  if (intersections mod 2)=1 then return true
  otherwise return false
end function
```

除此之外，还存在其他方法可处理光线与顶点是否切向相交。其中，最为简单的方法是，针对连接当前顶点的两个邻接点的直线段，检测其与光线的相交状态。如图 10.8 所示，若交点位于直线段内部，则光线与顶点切向相交。若多边形的顶点数量较少，则该方法较为简单，且一般不会超过两轮计算。

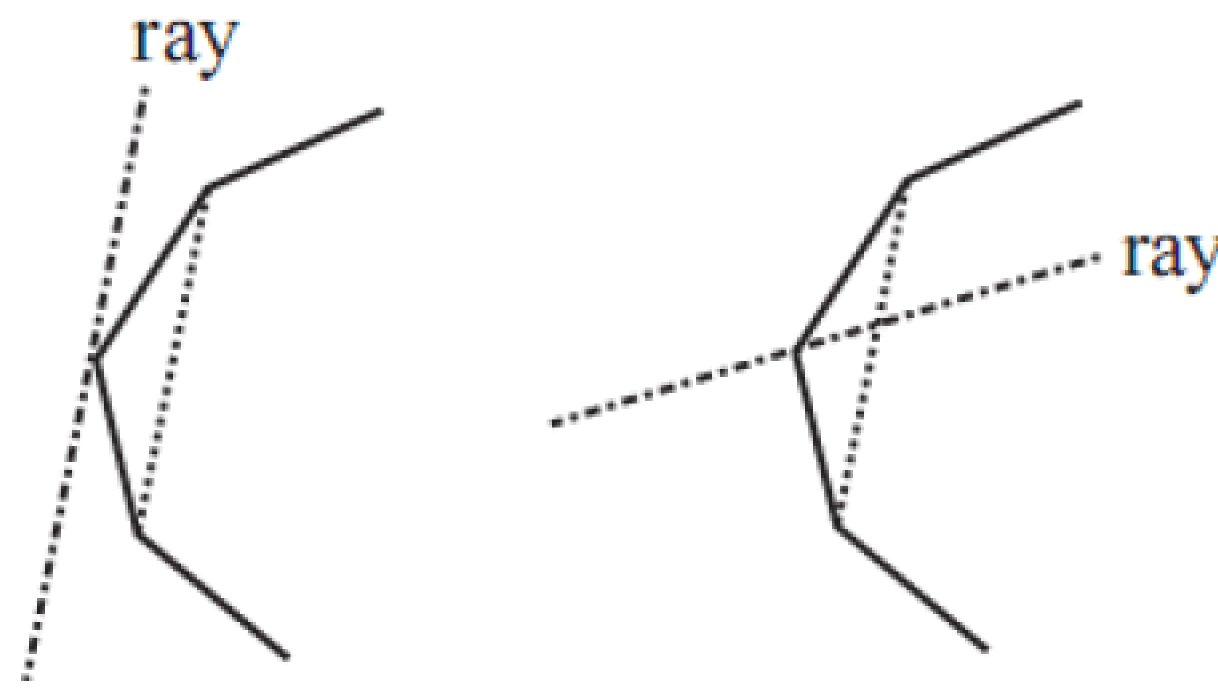


图 10.8　确定基于顶点的相交类型

10.3　某些合理性问题

任意形状之间的碰撞检测通常较为耗时，对此，有必要回顾一下某些标准的处理方案，此类方案提供了改进措施的参考标准。

10.3.1　计算复杂形状的前缘边

若类似于果冻状的几何形状 S 以恒定向量向墙壁运动，则预先计算其相对于墙面的前缘点将节省大量的计算时间。换而言之，只需预计算 S 的边界上的首个顶点即可，如图 10.9 所示。当然，还可能存在多个前缘顶点且与墙面于同一时刻碰撞，此处仅需计算一个数据点。

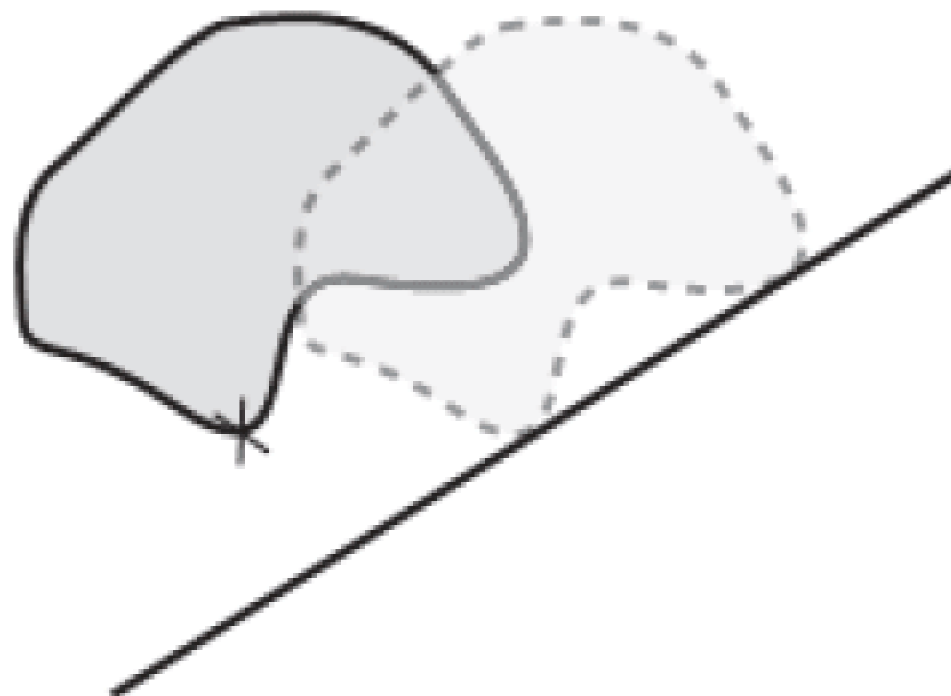


图 10.9　相对于墙面的复杂形状的前缘点

针对基于位图描述的图像，计算前缘点最简单的方式是检测全部数据点，并计算特定方向上

的首个顶点。对于多边形而言，该过程相对简单。同样，基于多边形的相关方案一般也适用于其他采用函数方式描述的对象。

这里，假设 S 表示为一个多边形，当使用与圆形相同的方法时，可计算各顶点在运动方向上与墙面之间的距离。对此，前缘点即为最近点。leadingPointOfPolygon()函数实现了这一方案，如下所示：

```
function leadingPointOfPolygon(poly, vel, wallPt, wallVect)
  set min to -1
  set minpt to 0
  //calculate the normal to the wall
  set n to norm(normal(wallVect))
  if dotProduct(n,vel)<0 then set n to -n
  repeat for each pt in poly
  set c to component(wallPt-pt, n)
  if c<0 then return "past"
  if min=-1 or c<min then
    set min to c
    set minpt to pt
  end if
  end repeat
  return pt
end function
```

虽然 leadingPointOfPolygon()函数包含多种用途，但若可计算前缘边，则该函数将更为有效。其中，前缘边表示为边界上的点集，并与墙面或其他静态对象方式碰撞。读者可回顾该碰撞方式于矩形之间的计算过程：针对特定的运动方向，至少存在一个矩形点未与其他对象碰撞。

与前缘点相比，前缘边的计算过程稍显复杂。其中，顶点不再是首要考察目标，相反，需针对各边加以分析。为了对此予以描述，假设当前几何形状与粒子碰撞而非墙面。相应地，该粒子可与边上任何位置方式碰撞。

根据惯例，假设全部顶点以逆时针方向排列，如图 10.10 所示。据此，内部边表示为连接两个连续顶点的逆时针边。在图 10.10 中，可将顺时针边法线作为外向法线。

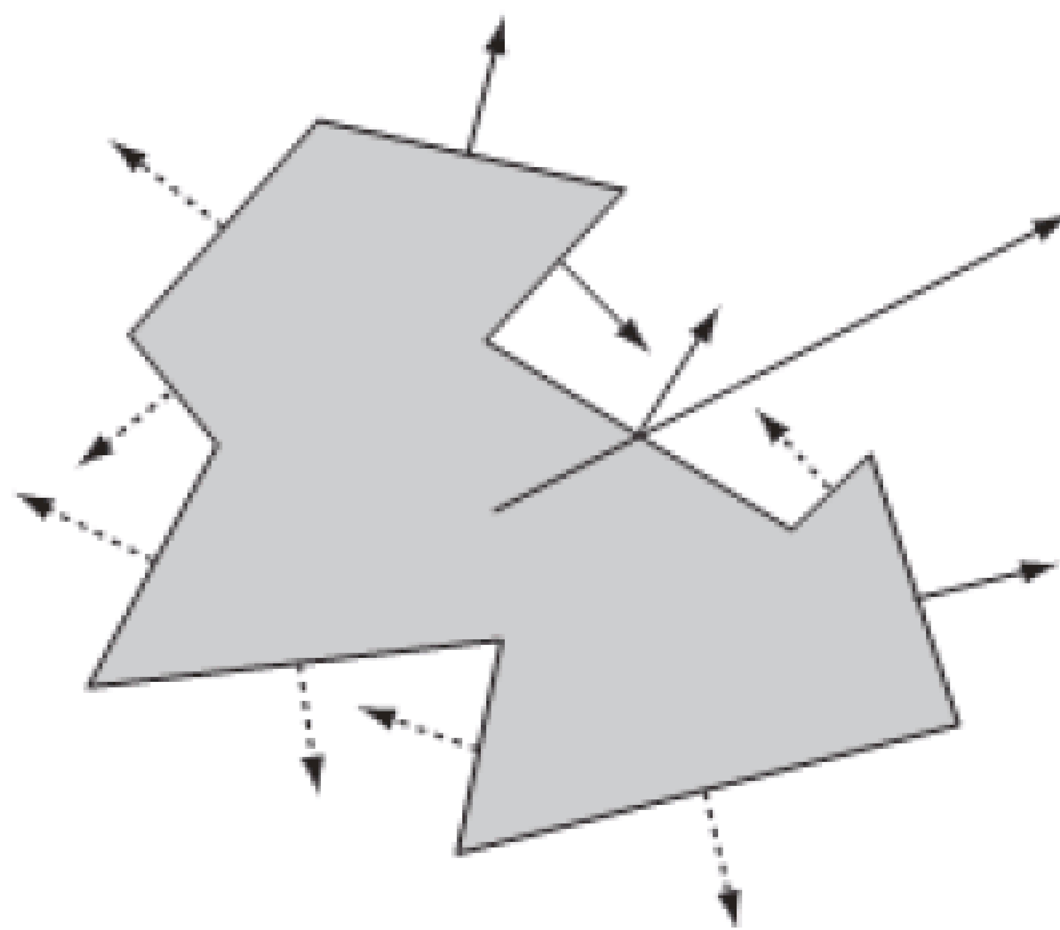


图 10.10　计算多边形的前缘边

进一步讲，令当前多边形沿向量 **v** 方向运动，因而碰撞边可描述为，内部边背离向量 **v**。也

就是说，边法线与 **v** 之间的点积为正值，在图 10.10 中，此类边采用实箭头标注。

leadingEdgeOfPolygon()函数实现了这一过程，如下所示：

```
function leadingEdgeOfPolygon(poly, vel)
  set edges to an empty array
  set c to the number of points in poly
  repeat with i=1 to c
    set v to poly[(i mod c)+1]-poly[i]
    if dotProduct(clockwiseNormal(v),vel)<0 then
      append array(poly[i],v) to edges
    end if
  end repeat
  return edges
end function
```

【提示】 *leadingEdgeOfPolygon()函数通常会使用到 clockwiseNormal()函数，其定义将在第 13 章加以讨论。*

待前缘边计算完毕后，碰撞检测过程将得到极大简化。实际上，其任务量约减少了一半。为了检测与 S 之间的碰撞行为，此处仅需要计算与任意目标直线段之间的首次碰撞。

鉴于目标直线段已知，因而当前计算可演变为前缘点，特别是基于函数形状的前缘点。对于多边形而言，相对于墙面的任意形状；前缘点即为距墙面的最近点。该过程涉及微积分运算，且通常情况下较为复杂。然而，若设置相关限制条件，则问题可得到一定简化。

由于大多数形状均通过参数函数加以表达，因而针对函数 x 和 y 以及 t 值，该形状上的各点等价于$(x(t), y(t))$。例如，在圆形中，对应函数定义为 $x(t) = r\cos(t)$，$y(t) = r\sin(t)$。当确定距墙面的最近点时，需要计算二者间的最小值。如前所述，可首先确定适当方向上的墙面法线$(n_1\ n_2)^{\mathrm{T}}$；随后，还需计算相对于墙面参考点$(p_1\ \ p_2)^{\mathrm{T}}$ 的任意边界点的法线分量。下列算式显示了基于该任务的一种计算方案：

$$D(t) = n_1(p_1 - x(t)) + n_2\,(p_2 - y(t))$$

当计算 D 的最小值时，须根据 t 执行微分计算，如下所示：

$$D(t) = -n_1\dot{x}(t) - n_2\dot{y}(t)$$

需要注意的是，体现参考点的对应常量不复存在——前缘点不应受到与墙面距离的影响。

函数 $D(t)$的最小值应位于 $\dot{D}(t)$ 的 0 值处，该值与 x 和 y 函数有关。例如，假设 x 和 y 定义为圆形，则可得到如下方程：

$$\begin{aligned}\dot{x}(t) &= -r\sin(t)\\ \dot{y}(t) &= -r\cos(t)\end{aligned}$$

最终，$\dot{D}(t) = n_2 r\cos(t) = n_1 r\sin(t)$。

若 **n** 表示为单位向量，针对某一 s 值，上述结果还等价于$(\sin(s), \cos(s))$，这也意味着，当且仅当 $\sin(s)\cos(t) - \cos(s)\sin(t) = 0$（消除常量因子 r），$\dot{D}(t) = 0$。通过三角恒等式，这等价于 $\sin(s-t) = 0$，对应结果表示为 $s-t = 0$ 或 $s-t = \pi$（或 π 的倍数）。该结果表明，前缘点可表示为与法线向量具有相同角度的点，或完全方向的数据点。

10.3.2　使用碰撞图

多数时候，对象位图可视为唯一操作方式，这也是碰撞图的用武之地。其中，碰撞图包含两种形式，即全碰撞图和高度图。高度图可视为某一方向上的碰撞图，且常用于地形处理中。作为一类通用规则，高度图在不同高度处表示为不同的单一直线。若高度图不包含悬垂（overhang）部分，则可通过单值函数予以生成，例如 $y(x)$。

通常情况下，并不需要使用到简单的高度图或碰撞图，此类数据图仅告知形状边界的所处位置，且不包含与点法线相关的重要信息。待碰撞或高度图计算完毕后，还需要创建第 2 幅图像以存储边界上各点的法线信息。更为重要的是，针对各边界点，读者可将此类信息编码至独立的碰撞图中。

上述信息的收集过程颇具技巧。由于对应信息采用逐像素方式加以组织，而非向量。因而难以获取任意点处的准确的切向信息。例如，图 10.11 显示了平滑对象经放大后的碰撞图，通过观察可知，任意标记直线均为特定碰撞点处的正确切线，其中还有包含相应切线的 3 条曲线。

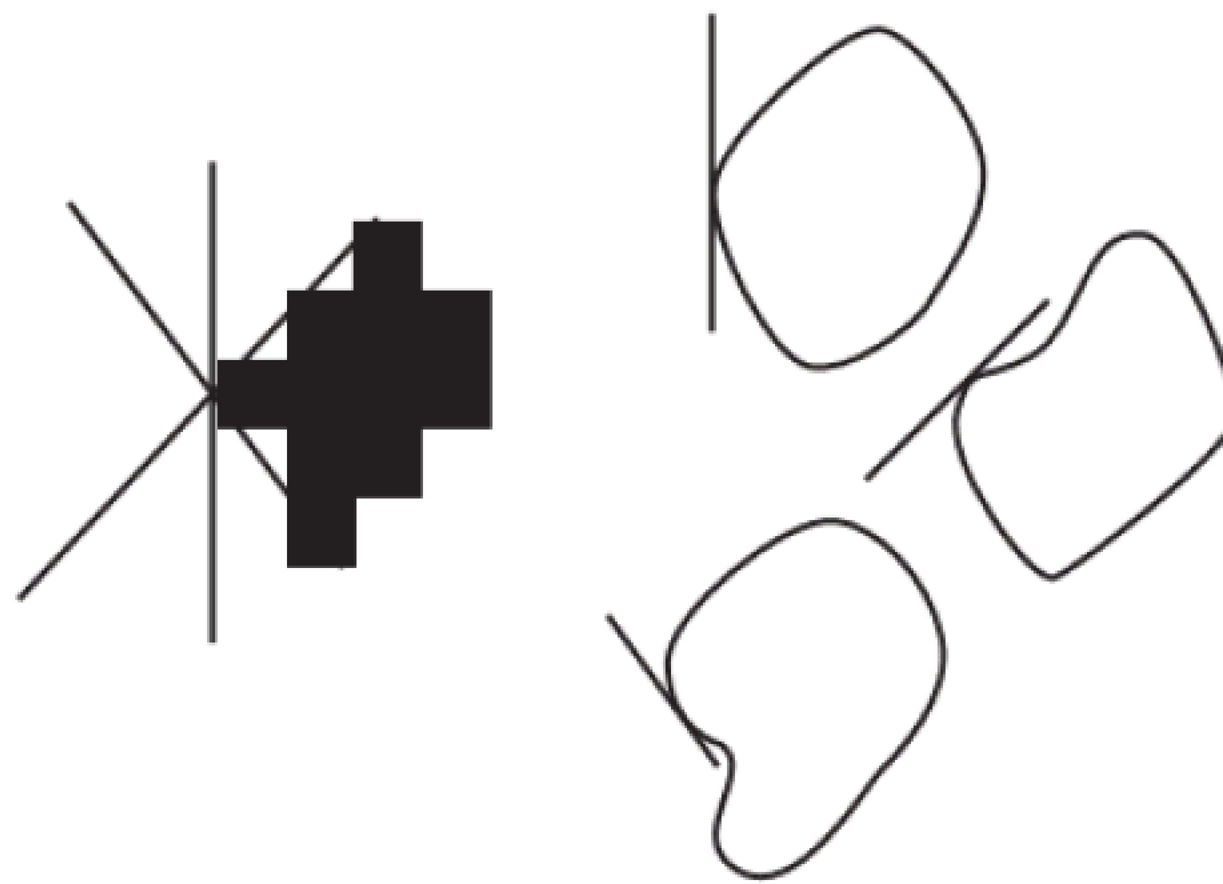


图 10.11　计算碰撞图切线时的问题

上述方案的猜测过程并非空穴来风，一如 calculateNormals()函数的实现过程。若计算了点 P 两侧邻接像素之间的梯度值，则可对其执行均值计算，进而猜测点 P 处正确的梯度数据。然而，该方案依然与几何形状的平滑性有关。与碰撞对象相比，凹凸和曲线应相对平滑。

开始阶段，读者可尝试使用高度图，其计算过程相对简单，如下所示：

```
function calculateNormals(heightMap)
  set normals to an empty array
  repeat with i=1 to the number of elements of heightMap-1
    set hd to heightMap[i+1]-heightMap[i]
    set v to norm(-hd, 1)
    if i=1 then
      set thisNormal to v
    otherwise
      set thisNormal to (v+lastNormal)/2
    end if
    append thisNormal to normals
```

```
    set lastNormal to v
  end repeat
  append lastNormal to normals
  return normals
end function
```

对于更为通用的碰撞图，其检测过程也更具技巧性。对此，假设相应的碰撞图为黑白图像，因而需要跟踪此类图像的边数据。这里，各像素包含 3 种可能值，分别表示为内部的黑色、外部的白色以及边处的灰色。作为直接邻接数据，边像素中的黑色像素中包含了白色像素。findEdges()函数提供了基本实现方案，如下所示：

```
function findEdges(bwImage)
  set newImage to a copy of bwImage
  repeat for each point in bwImage
  if the point is black then
    if at least one neighbor of the point is white then
      set the equivalent point in newImage to gray
    end if
  end if
  end repeat
  return newImage
end function
```

findEdges()函数使用了包含灰色轮廓线的图像。

待此类数据图处理完毕后，法线的计算过程则与前述内容相同。其中，一种较为方便的方法是将法线信息保存至碰撞图自身。特别地，此处可不采用黑色、白色和灰色数据，相反可使灰色量值根据法线角度尝试变化。若图像每个像素包含 8 位数据（即灰度图），则针对各边像素包含 254 个可能值；若涉及内部像素的空像素以及 0 值，则包含 255 种可能值。因此，如果某一像素的数据值为 200，则法线角度值表示为 $200\times\dfrac{2\pi}{254}$。calculateNormals()函数显示了具体的实现过程，如下所示：

```
function calculateNormals(collisionMap)
  set newImage to a copy of collisionMap
  repeat for each point in collisionMap
    if the color of the point is white or black then next repeat
    set clist to an empty array
    set neigh to 0
    repeat for neighbor in (0,1), (1,0), (0,-1),(-1,0)
      if the color of point+neighbor is white then
        append 0 to clist
      otherwise
        append 1 to clist
        add 1 to neigh
      end if
    end repeat
    if neigh=0 then set v to 0
    if neigh=1 then set v to -(the neighbor vector)
```

```
    if neigh=2 then
      if clist[1]=1 and clist[3]=1 then set v to (1,0)
      if clist[2]=1 and clist[4]=1 then set v to (0,1)
    otherwise set v to -(the average of the neighbor vectors)
    if neigh=3 then set v to the non-neighbor vector
    set v to norm(v)
    set the color of the corresponding
        point of newImage to writeColor(v)
  end repeat
  return newImage
end function
```

当处理颜色值时，可调用 writeColor()函数，该函数定义如下所示：

```
function writeColor(v)
  set a to atan(v[2],v[1])
  return integer(a*127/pi)
end function
```

与颜色值写入相对应的是颜色值的读取操作，对应函数为 readColor()，其定义方式如下所示：

```
function readColor(c)
  set a to c*pi/127
  return vector(cos(a),sin(a))
end function
```

【提示】calculateNormals()函数的实现过程相对粗糙，其可能的法线向量仅为 45° 的倍数。在练习 10.2 中，读者将尝试生成连续的法线数据。

待基于法线的碰撞图实现完毕后，则可计算碰撞图与运动粒子之间的相交结果。由于高度图相对简单，因而在开始阶段可使用此类数据。这里，假设粒子位于点 P 处，且在高度图 H 中距地面位移为 **s**，即假设粒子始于 $x = 1$ 处。同时，粒子在运动过程中 x 坐标在 p_1～p_1+s_1 之间变化。若任意 x 值处，y 坐标大于或等于高度图坐标（向下计算），则粒子将产生碰撞。particleHmapCollision()函数实现了这一过程，如下所示：

```
function particleHmapCollision(p, s, h)
  if s[1]=0 then
    //vertical motion:only one check
    if h[p[1]]<=p[2]+s[2] then
      return (h[p[1]]-p[2])/s[2]
    otherwise
      return "no collision"
    end if
  otherwise
  set grad to s[2]/s[1]
  repeat for i=0 to s[1]
    set x to p[1]+i
    set y to p[2]+grad*i
    if h[x]<=y then
```

```
      return s[1]/i
    end if
  end repeat
  return "no collision"
  end if
end function
```

通过观察可知，particleHmapCollision()函数并未返回法线数据，该数据可在法线图的帮助下得以实现。

相同的处理过程也适用于通用的碰撞图，除了测试大于高度图坐标的数据之外，还需进一步检测粒子是否位于几何形状内部。由于内部点采用黑色予以标记，因而该处理步骤并不复杂。pointCmapIntersection()函数使用了位于位置 Q 处的某一形状以及碰撞图 C，如下所示：

```
function pointCmapIntersection(p, s, q, c)
  set d to magnitude(s)
  if d=0 then return "no intersection"
  set sn to s/d
  set st to p-q
  repeat with i=1 to integer(d)
    set pos to st+i*sn
    set col to the color of the pixel in c at pos
    if col is not black then return d/i
  end repeat
end function
```

虽然上述函数工作良好，但依然可通过多种方式对其加以改进，其中的一个原因即是计算速度较慢。类似地，若需要计算法线数据，则该函数仍然需要予以修正。若像素颜色为黑色，则粒子将越过当前边。然而，由于仅检测运动过程中的像素长度子步骤，因而边像素最多为 1 个像素。除此之外，此处计算应采用整型数据予以执行。总体而言，Bresenham 算法涵盖了上述计算目标，第 22 章将对此加以讨论。相应地，另一种优化方案则可避免大范围地对运动行为进行检测，相反，该优化方案仅查看端点。仅当端点位于当前几何形状内部时，方计算准确的交点信息。另外，相对于当前几何形状的尺寸和不规则性，仅当粒子的位移较小时，该方案方趋于稳定。

另一种加速方案称作碰撞环（collision halo）。其中，几何形状外部区域不采用纯白色数据，并通过灰色着色标记与几何形状最近边点的距离。通过测试该值，可计算基于特定位置的碰撞行为。

若碰撞对象大于点粒子，则碰撞计算将更加冗长。针对不规则几何形状，相对于运动对象前缘边的碰撞测试并不存在较好的替代方案。取决于碰撞对象的规则程度，读者可采用适当的简便方案予以计算；然而，对于某些复杂的场景，其处理过程依然十分复杂。

10.3.3　计算包围形状

一种可加速碰撞检测计算的方法称作包围形状，前述碰撞图隐式地实现了这一方案。包围形状表示为包围碰撞对象的简单形状，例如，若对人物角色执行碰撞测试，可使用矩形包围该角色；而对于前述海星形状，则可使用圆形。

当使用包围形状时，在分析细节内容并检测精确的像素级别碰撞之前，可预计算两个对象之间是否产生碰撞。高效的包围形状应体现应有的简单性和最大包围性，例如，灯柱的包围圆超出了实际的碰撞范围，而狭长矩形则更为紧凑。

作为一类通用规则，应事先确定包围形状，例如矩形、圆形、三角形或其他多边形。当包围形状确定完毕后，还需要进一步计算包围形状的尺寸。与多边形相比，圆形则提供了相对完备的选择数据。若多边形包含顶点 $\mathbf{p}_1$，$\mathbf{p}_2$，…，$\mathbf{p}_n$，则中心位置位于顶点平均值处，即$\frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i$。不难发现，这也是当前形状的质心位置。据此，boundingCircle()函数的全部工作即是计算顶点距中心位置的最大距离（此处为圆半径），如下所示：

```
function boundingCircle(poly)
  set c to the number of vertices in poly
  set s to (0,0)
  repeat for each v in poly
    add v to s
  end repeat
  set center to v/c
  set mx to 0
  repeat for each v in poly
    set d to magnitude(v-center)
    if d>mx then set mx to d
  end repeat
  return array(center, mx)
end
```

boundingCircle()函数并未计算最小圆，当针对较为规则的形状，该函数已然足够。较好的算法将计算最小的包围圆，其他形状也存在类似的情况。

相比之下，包围盒则稍显复杂。这里，计算某一特定轴向上的包围盒并不困难，但对于全部轴向，情况则复杂得多。对此，轴对齐包围盒（AABB）和对象对齐包围盒（OABB）可视为两种较为常见的选项。在 AABB 中，盒体沿主轴对齐，通常为 x 轴和 y 轴。在三维环境中，主轴还涉及 z 轴。

若模拟环境中的全部对象均包含于同一轴对齐的包围盒，则包围盒之间的碰撞检测可得到适当简化。相应地，可通过考察 x 和 y 分量测试碰撞行为，并可在更为通用的算法中移除大量的点积计算。另外一方面，若对象形状无法满足 AABB 的紧密拟合，或者，若对象旋转使得 AABB 在一段时间内产生变化，则简化效果将会大大降低。

与 AABB 相比，OABB 则更具优势。其中，轴向根据当前形状的拟合程度予以选择，且无须担心轴向的方向问题。尽管这使得碰撞检测过程趋于复杂化，但该方案更为通用，且无须在对象旋转时重计算包围盒。

图 10.12 分别显示了 AABB 和 OABB，在左图中，对应形状被与 x 轴和 y 轴对齐的 AABB 所包围。不难发现，该矩形过于庞大且无法较好地反映出当前几何形状。相比之下，右图则选取了一对较好的轴向，对应的 OABBKiev 紧密地包围当前形状。

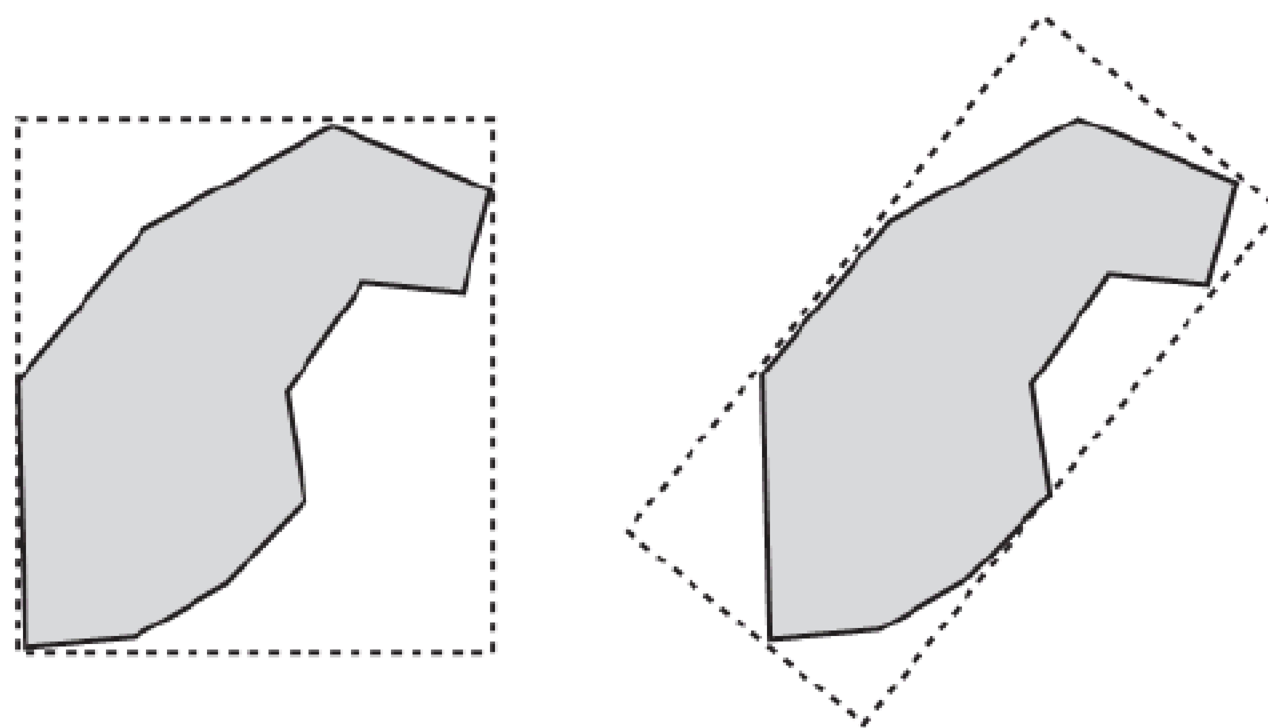


图 10.12　同一形状的 AABB 和 OABB

计算最佳 OABB 可视为一类颇具技巧的数学问题，且与因子分析有几分类似。对此，需计算距顶点最大距离和最小距离的轴向，即最佳匹配直线。通常情况下，可通过最小二乘法对此进行计算。也就是说，当前任务为计算一条直线，且全部顶点的平方垂直距离保持最小。虽然最小化绝对值常可得到较好的计算结果，但却难以通过分析方式予以实现。这将演变为当前矩形的长轴，且短轴与其保持垂直状态，对应的半长值为最大距离。同样，类似方案也适用于三维环境。

最小二乘法相对直观，此处假设数据点列表定义为$(x_1, y_1), (x_2, y_2)$，…，(x_n, y_n)，随后可计算 x 和 y 变量的均值结果。除此之外，还需进一步计算变量的方差，即距均值的偏离程度。通常，方差定义为标准偏差的平方值，且数据集的方差按照下列方式予以确定：

$$v_x = \frac{1}{n}\sum_{i=1}^{n}(x_i^2) = \overline{x}^2$$

【提示】上述方差公式使用了大写希腊符号∑，其上方分别标注了“$i = 1$”和 n，表示后续数据值之和。其中，索引 i 通过 1～n 进行替换。

待方差计算完毕后，则可计算量值 S，如下所示：

$$S = \frac{n(v_x v_y)}{\sum_{i=1}^{n} x_i y_i - n\overline{xy}}$$

其中，分母表示为方差的二维版本，并可称作 v_{xy}。

根据上述信息，最佳匹配直线可通过 $y=ax+b$ 加以确定。其中，$a=-2S\pm\sqrt{4S^2-1}$ 且 $b=\overline{y}-a\overline{x}$ 。

通过上述方案可定义 lineOfBestFit()函数，另外，该函数使用了两个其他函数分别计算均值和方差值，如下所示：

```
function lineOfBestFit(dataPoints)
  set xlist to arrayOfValues(dataPoints,1)
  set ylist to arrayOfValues(dataPoints,2)
  set n to the number of elements in dataPoints
  set mx to mean(xlist)
  set my to mean(ylist)
  set sx to variance(xlist)
  set sy to variance(ylist)
  set sxy to variance(xlist,ylist)
  if sxy=0 then return "vertical"
  set s to n*(sx-sy)/sxy
```

```
  set a to -2*s+sqrt(4*s*s-1)
  set b to my-a*mx
  return array(a,b)
end function
```

如前所述，mean()函数如下所示：

```
function mean(list)
  set s to 0
  set n to the number of elements in list
  repeat for i=1 to n
    add list[i] to s
  end repeat
  return s/n
end function
```

variance()函数使用了 mean()函数，其实现内容如下所示：

```
function variance(list,list2)
  if list2 is undefined then set list2 to list
  set m1 to mean(list)
  set m2 to mean(list2)
  set s to 0
  set n to the number of elements in list
  repeat for i=1 to n
    add list[i]*list2[i] to s
  end repeat
  return s/n-m1*m2
end function
```

待最佳匹配直线计算完毕后，可将其转换为向量形式，进而生成当前几何形状的方向和位置，而垂直向量可生成短轴。随后，可根据各轴向计算顶点的最大距离，并可得到 OABB 各边的长度。

一旦设定了包围形状，碰撞检测技术则无太多变化。针对物理学而言，读者可将某些基本形状视为包围形状。当存在潜在碰撞时，则可切换至全碰撞状态；或者，取决于当前形状及其包围形状之间的匹配程度，还可跳过全碰撞这一步骤。若某一对象接近于圆形，读者可直接将其视为圆形形状。

10.4　内 建 方 案

本节内容相对独立，且涉及较少数学内容。一种可能的情况是，可在程序设计语言中使用内建函数处理任意形状。例如，某些语言提供了 intersects()函数，虽然这一类函数的完善性有待提高，但某些时候，它们依然不失为最佳方案。

需要注意的是，此类函数均为本章前述内容所讨论的不同版本的计算函数。若读者正使用位

图应用程序协同工作，此类函数将使用碰撞图系统；若与基于向量的应用程序协同工作，此类函数将使用基于向量的引擎。其优点不一而同，但缺点却基本相同，这主要体现在，几何形状越复杂，计算机的计算能力也会随之下降。

一种解决方案是使用碰撞代理机制，该机制定义了某一几何形状，进而提供了一类简化的小型碰撞对象版本，例如，本章开始处，当生成碰撞图时即采用了碰撞代理。另外，从某种程度上讲，当生成包围形状时同样使用了代理机制。总体而言，针对自定义级别的碰撞方向，碰撞代理可视为一类有效机制。据此，真实的碰撞对象从属于代理对象，也就是说，实际物理行为发生于代理对象上，而真实对象则处于附属地位，引擎的全部工作则是计算代理对象的碰撞行为。

10.5 本 章 练 习

【练习 10.1】试编写 splitPolygon(poly)函数，该函数接收顶点呈顺时针排列的任意多边形，并将其划分为三角形。

该函数并非想象中的简单，对此，可使用递归函数计算 3 个邻接角点，并据此构成一个三角形。针对通用多边形，应确保三角形不与其他边相交。

【练习 10.2】试编写 smoothNormals(collisionMap)函数，该函数使用 calculateNormals()函数生成的碰撞图，并生成一个包含真实法线的新数据图。

该函数可使用与高度图类似的系统，并对邻接边顶点之间的法线向量执行“均值”计算。除此之外，读者还可尝试实现本章前述内容所讨论的碰撞环。

10.6 本 章 小 结

本章讨论了多种概念，并展示了碰撞检测的综合应用技术。当然，与碰撞检测相关的内容远非如此。读者应确保理解本章所阐述的基础内容，并为后续学习打下坚实的基础。

第 11 章将运用前述所学知识制作一款游戏。

至此，读者应掌握如下内容：

- 如何通过位图或向量描述通用几何形状。
- 如何通过 Bezier 或 Catmull-Rom 样条定义曲线。
- 术语“凸形”和“凹形”的含义，以及凹多边形的识别方式。
- 如何检测与任意多边形之间的碰撞行为。
- 当与墙面碰撞时，如何计算任意形状的前缘点，以及多边形的前缘边。
- 如何计算碰撞图或高度图（包括边法线），以及粒子与二者间的碰撞行为。
- 如何计算包围圆或 2D 对象对齐包围盒（OABB）。
- 如何使用代理机制加速碰撞检测过程。

第 11 章　一款简单的撞球游戏

本章包含如下内容：

- 概述。
- 模拟中的主要元素。
- 运行游戏。

11.1　概　　述

在前述章节的基础上，本章将继续讨论各类模拟环境以及基于真实物理的游戏。通过基本的碰撞处理方案，读者可尝试制作撞球游戏、弹珠游戏、乒乓球游戏以及打砖块游戏。截止到目前为止，相信读者已可处理大多数相对复杂的碰撞事件。

本章将运用前述知识制作一款标准的撞球游戏，需要说明的是，游戏的制作过程不同于函数的编写。函数可将数学运算独立开来，或以伪代码方式编写算法。而游戏的制作过程则需要关注特定的编程环境细节。再次强调，对应示例代码体现了应有的通用性以及完整性，读者应对其予以适当改写以使其适应当前编程环境。类似地，游戏制作过程还包含大量的游戏逻辑细节内容，例如制定游戏规则、跟踪玩家，或判断游戏的胜负结果。本章暂不讨论此类细节内容，而是将重点移至数学领域。与此同时，编程环境也将得到进一步扩展，进而获得一类更为通用、有效的应用环境。

11.2　模拟中的主要元素

撞球游戏可归于真实模拟范畴内，对此，首要任务即是构建模拟框架，这涉及模拟的工作环境及其限制条件。对此，须先期考察当前环境与期望目标之间的关系。多数时候，期望目标往往会受到开发工具的限制；而某些时候，借助于开发工具，常可获得意外的结果。在撞球游戏示例中，开发过程始于撞球桌面，其定义方式决定了球体在桌面上的运动行为，而球体的运动方式则通过球体与桌面间的物理参数加以定义。

11.2.1　定义撞球桌面

关于撞球游戏所涉及的函数，大量的约束条件也变得越发明显，例如前述章节所讨论的 2D

碰撞问题。对此，应首先正确地描述撞球游戏的桌面。图 11.1 显示了水平桌面的俯视图，且表示为一个矩形。这里，存在多种方式可定义桌面，其中最为简单的方法是将其划分为 6 个主要元素，且各元素表示为一条直线段。此处，桌面的落袋直线段未覆盖的矩形边。

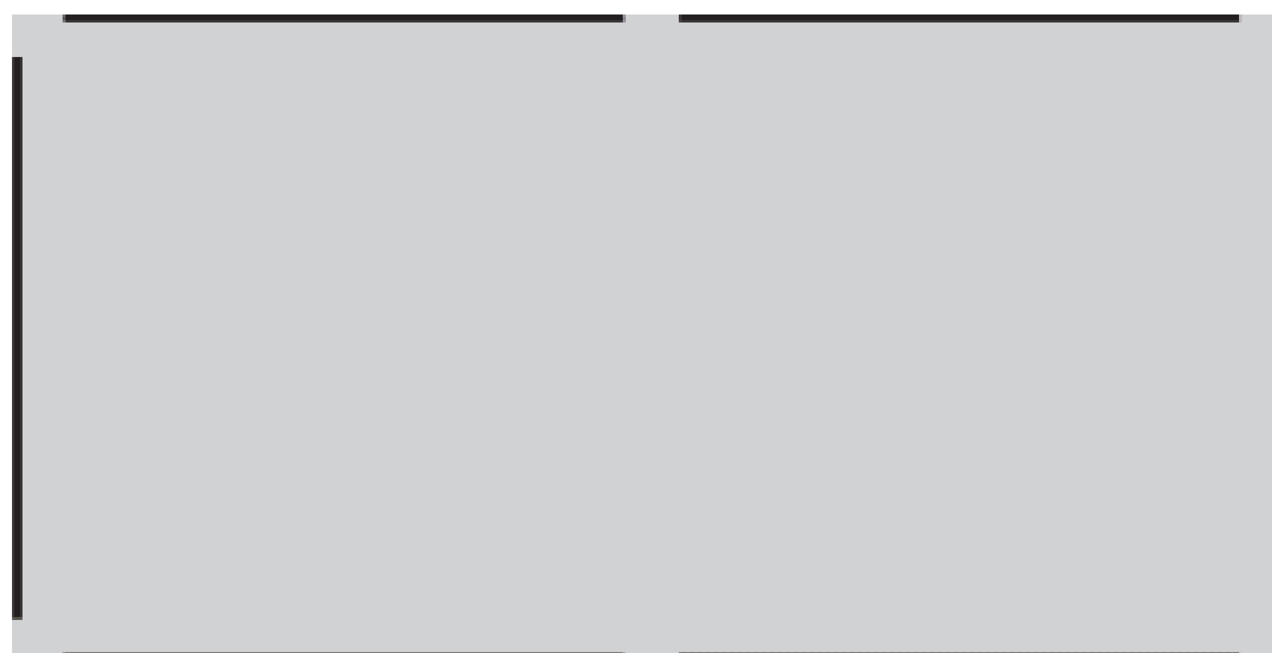


图 11.1　定义撞球桌面

球体与桌面四壁碰撞或者彼此碰撞，此类碰撞可视为弹性碰撞。类似地，由于桌面水平放置，因而重力可不予考虑。除了摩擦力和母球撞击时刻，其他情况下球体均不存在加速行为。

关于球体的落袋方式，可考察图 11.2 所示的袋口，其中，各袋口壁表示为 1/4 圆弧，即图中未填充的圆周。相对于桌面的滚动表面，由于另外两个袋口壁通过两个内部圆弧予以连接，因而两个圆的大部分圆周对于碰撞检测行为不可见。当袋口壁定位完毕后，当球体穿越圆弧时（即填充圆弧），则该球体标记为落袋。图中，6 条直线段构成了库边，12 个顶点构成了台口，因而球体和桌面之间存在 18 种潜在碰撞。

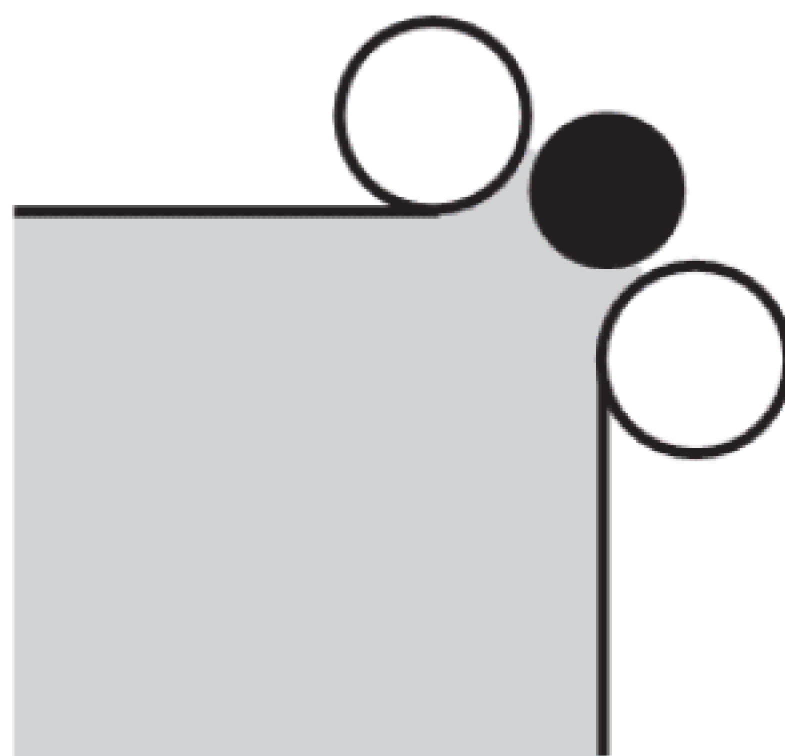


图 11.2　袋口处的细节内容

针对桌面和袋口尺寸，须考察 4 点内容。首先是桌面尺寸，其次是围绕落袋呈曲线状态的台口，然后是台口之间的空隙，最后则是球体的尺寸。关于尺寸数据，球体可表示为一个矩形，由于该矩形 2 倍于球体宽度，因而可定义为两个正方形。针对台口和落袋之间的空隙，其目的在于使得台口稍大于球体半径，但不可过大。对此，可适当增加或减少台口圆弧的半径值。在当前示例中，台口半径设置为 1/2 球体半径。对于袋口尺寸，可将其设置为 1.7 倍的球体宽度，这也意味着，球体可通过的空隙 1.2 倍于球体的宽度值，且稍大于真实撞球桌面的空隙值。

桌面的视觉化描述涉及大量的工作，当对此定义相关函数时，与撞球桌面、落袋、库边以及球体尺寸均应通过通用变量加以定义。据此，若读者期望调整参数（增加或降低游戏难度），则无须遍历全部代码进而修改硬编码数据值。

defineTable()函数负责设定撞球桌面的主要属性值，该函数接收球体尺寸、桌面尺寸、落袋

尺寸以及落袋的台口尺寸作为参数。在实际应用过程中，该函数可能稍显复杂。例如，当绘制桌面库边时，可能需要对桌面稍作偏移。在当前示例中，桌面的左上角定义为(0,0)。对应函数如下所示：

```
function defineTable(ballRadius, tableSize, pocketSize, jawSize)
  set rs to ballRadius*pocketSize
  set rd to ballRadius*sqrt(2.0)*pocketSize //just for convenience
  set walls to an empty array
  append these arrays to walls:
   ((rd,0), (tableSize-rd*2,0)) //top
   ((0,rd), (0,tableSize-rd-rs)) //left top
   ((0,tableSize+rs), (0,tableSize-rd-rs)) //left bottom
   ((rd,tableSize*2), (tableSize-rd*2,0)) //bottom
   ((tableSize,rd), (0,tableSize-rd-rs)) //right top
   ((tableSize,tableSize+rs), (0,tableSize-rd-rs)) //right bottom
  set pw to ballRadius*jawSize
  set jaws to an empty array
  //you now use walls as a guide to draw jaws
  repeat for wall in walls
    if wall[2][1]=0 then //this is a vertical wall
      if wall[1][1]>psize/2 then //it's on the right
        append (wall[1]+(pw,0)) to jaws
        append (wall[1]+wall[2]+(pw,0)) to jaws
      else //it's on the left
        append (wall[1]-(pw,0)) to jaws
        append (wall[1]+wall[2]-(pw,0)) to jaws
      end if
    else //this is a horizontal wall
      if wl[1][2]>psize then //it's on the bottom
        append (wall[1]+(0,pw)) to jaws
        append (wall[1]+wall[2]+(0,pw)) to jaws
      else //it's on the top
        append (wall[1]-(0,pw)) to jaws
        append (wall[1]+wall[2]-(0,pw)) to jaws
      end if
    end if
  end repeat
  return array(walls,jaws)
end function
```

11.2.2　定义球体

撞球桌面表面由多条开球线加以划分。在真实的撞球游戏中，当玩家准备开球时，可将母球置于开球线上的任意位置，在当前示例中，出于简单的考量，母球可置于开球线上的同一位置。

在图 11.3 中，除母球之外的其他球体以三角形排列。其中，最前端球体位于桌面$\frac{3}{4}$处。图

中标记的直角三角形体现了基于球体半径比值的各距离值，进而展示了球体间的数学构造方式。

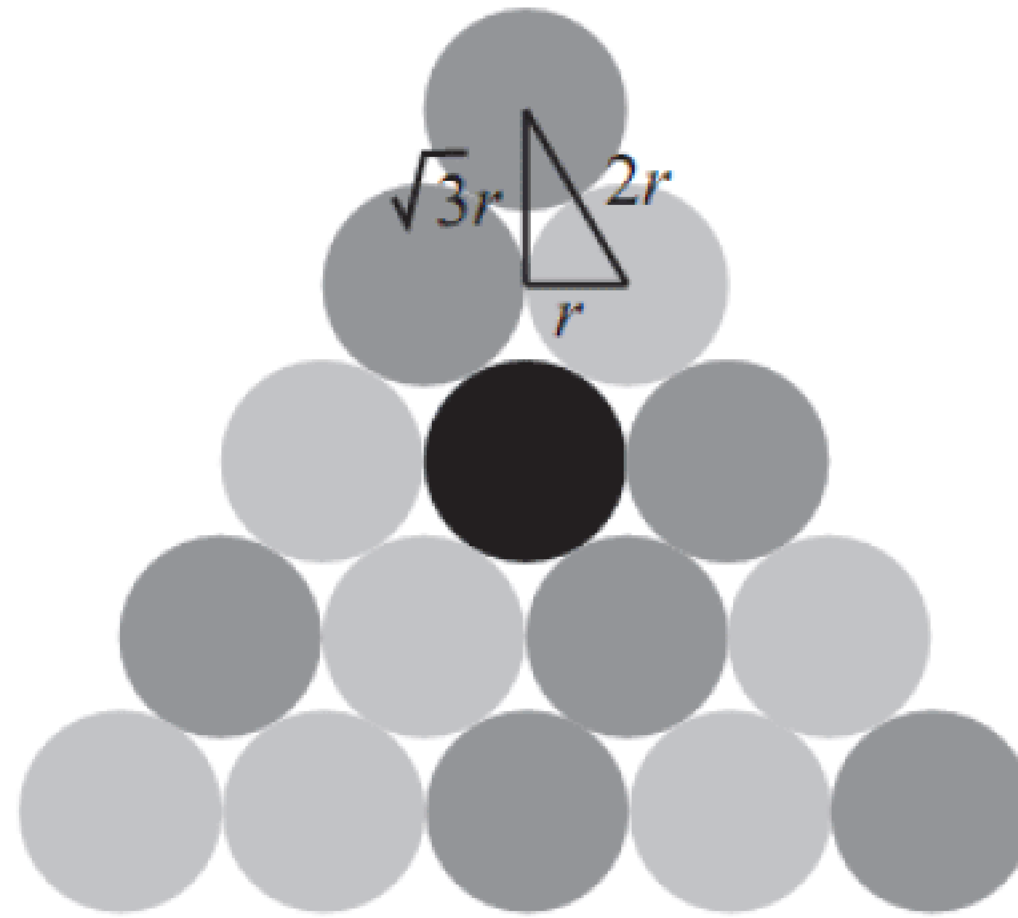


图 11.3　球体的初始状态

撞球游戏使用一个母球和 15 个目标球，为了表达这些球体对象，可使用包含 16 个数据元素的数组。其中，各数组元素包含了各个球体的信息。目前，球体数据信息包含球体的位置、方向、速度以及颜色。当描述球体的方向时，可对其使用单位向量；当速度为 0 时，对应方向为任意方向。

当在数组中设置球体信息时，可将母球信息置于首位，其他球体可据此依次加以排列，而黑球可置于最后。相应地，颜色顺序可表示为“cue”、“red”、“yellow”或“black”。其中，母球通常采用白色加以标识。

当构造三角形排列的球体时，球体间应处于分离状态。若球体间彼此接触，则舍入操作常会使球体处于交叠状态。createBalls()函数根据上述内容构造撞球桌面上的球体对象，该函数仅需要两个参数，即桌面的尺寸和球体的半径，如下所示：

```
function createBalls(tableSize, ballRadius)
  set r to ballRadius*1.02
  set cuestart to (tablesize/2,2*tablesize/5)
  set y to sqrt(3.0)*r
  set tristart to (tablesize/2,tablesize*3/2)
  set balls to an empty array
  repeat for i=1 to 16
    if i=1 then
      set col to "cue"
    else if i=16 then
      set col to "black"
    else if (i mod 2)=0 then
      set col to "red"
    else
      set col to "yellow"
    end if
    if i is
      1: set p to cuestart
      2: set p to tristart
      3: set p to tristart +(-r,y)
```

```
      4: set p to tristart +(r,y)
      5: set p to tristart +(2*r,2*y)
      6: set p to tristart +(-2*r,2*y)
      7: set p to tristart +(-3*r,3*y)
      8: set p to tristart +(-r,3*y)
      9: set p to tristart +(r,3*y)
      10: set p to tristart +(3*r,3*y)
      11: set p to tristart +(4*r,4*y)
      12: set p to tristart +(0,4*y)
      13: set p to tristart +(2*r,4*y)
      14: set p to tristart +(-4*r,4*y)
      15: set p to tristart +(-2*r,4*y)
      16: set p to tristart +(0,2*y)
    end if
    append array (p, 0,(1,0), col) to balls
  end repeat
  return balls
end function
```

尽管 createBalls()函数目前工作良好，但在后续的计算过程中，读者将会发现，各球体将使用到更多的属性信息。例如，球体是否落袋，或者球体是否处于运动状态（一类较为有用的记录信息）。

【提示】面向对象程序设计可方便地组织上述信息。对此，可针对球体对象定义类数据，各球体对象可跟踪自身的状态，并响应查询操作。然而，在当前环境下，为了清晰地表达主算法的意图，过程式方案则更为适宜。

11.2.3　定义物理参数

模拟过程中的最后一部分内容是定义物理参数，以使对象在游戏中保持一致状态。大多数数据往往通过反复试验这一方式加以确定，然而，在初始阶段，读者依然可采用某些近似方案。

由于撞球游戏的物理内容相对简单，因而多数参数于初始阶段即可确定。首先，全部球体对象均包含相同质量（包括母球），并呈现为完全弹性碰撞。最终，读者可使用第 9 章介绍的 resolveCollisionEqualMass()函数，回忆一下，该函数忽略了质量和效率值。

撞球桌面包含库边，这会对球体的碰撞方式产生影响。首先，球体间的碰撞相比，球体与库边之间的碰撞则相对低效，其结果为，无法充分地以弹性方式处理球体和库边之间的碰撞。

除了球体与库边之间的碰撞外，摩擦力则是另一个能量损失之处。摩擦力的真实处理十分复杂，且对游戏体验产生显著的影响。首先，若缺少摩擦力的支持，则球体无法滚动。为了有效地降低基于摩擦力的引擎复杂度，可尝试使用“游戏摩擦力”，且存在多种实现方式可生成相对真实的运动行为，如下所示：

- 每秒以恒定量值减少模拟能量。
- 每秒以恒定因子减少能量。
- 采用根据当前能量变化的因子减少能量。

对此，读者可能思考相应的最佳方案。其中，上述方案一将生成线性减少的速度结果（恒定减速运动）；方案二将以指数方式减速运动（根据速度实现减速变化）；方案三则位于前两者之间。尽管最后一个方案颇具真实感，但在实际操作中，该方案却与方案一无明显差别。另外，方案二可视为一类最差选择。针对指数递减，球体迅速减速，但会在较长时间内处于完全静止状态。实际效果可描述为，缓慢移动的对象即刻停止运动。

若采用第 3 种方案（该方案涉及可变速度），则应设置一个“钝化”参数。该参数旨在便于计算，并非与真实的物理行为相对应，并设置一个最小速度。如小于该速度，则对象视为处于停止状态。恒定减速运动（前两个方案）无须使用该参数，其中，全部球体对象可确保在合理的时间范围内低至 0 速度。

根据上述讨论，当前游戏示例的完整参数集如下所示：

- 桌面宽度。
- 球体半径。
- 落袋尺寸。
- 基于摩擦力的减速运动。
- 库边的碰撞效率。

上述参数值与游戏体验相关，进而生成某种程度的真实感。在真实游戏中，落袋的尺寸通过桌面宽度加以确定，然而，为了增加或降低游戏的难度，可适当地调整该值。另外，与难度和真实感相比，物理参数往往与游戏体感关联紧密。例如，此类参数可影响玩家的等待时间，以及动作的执行速度。另外，与库边的碰撞效率也会影响到斯诺克的难度。在斯诺克游戏中，为了获取合理的走位，玩家需要通过库边反弹母球。

11.3 运行游戏

当搭建起游戏的框架后，即可启动游戏，本小节将考察游戏的启动方式。首先，需要初始化主球的运动状态；随后，还需考察主游戏循环以及每一轮击球的开始方式；最后，还将模拟一个独立的游戏场景。

11.3.1 创建球杆

用户与游戏的交互方式与游戏的物理学和数学内容相关。在当前环境中，主要目标始于主球及其初始动量。在各次击球的开始时刻，球杆位于母球位置处。在常见方案中，球杆的击球点往往位于球体的外部区域中，球杆可根据鼠标实现旋转效果，并位于鼠标和球心之间的直线上。

当鼠标键按下时，球杆开始回撤，并在鼠标键释放时击打母球。此处，母球的初始动量与鼠标键按下的时间长度成正比。若鼠标键按下的时间超出了规定时间，则球杆自动以全力方式击打母球。

cueRotation()函数即采用了上述方案，该函数假设母球的位置已知，并可于特定时刻读取鼠

标位置。根据上述信息，即可确定母球和鼠标之间的向量。同时，将该向量转换为某一角度值则可设定球杆的旋转行为。除此之外，该函数的定义还与球杆的默认旋转有关。此处，假设球杆呈水平状态并指向右侧。随后，该函数可确定任意时刻的旋转状态，其定义方式如下所示：

```
function cueRotation(ballPos)
  set v to ballPos - (the current mouse position)
  if magnitude(v)>0 then
  set ang to atan(v[2],v[1])
  return ang*180/pi
  otherwise
  return "error"
  end if
end function
```

【提示】球杆的准确绘制取决于读者的开发平台。在某些场合下，这将涉及球杆的旋转属性。

当鼠标位于球体位置处时，cueRotation()函数可能存在潜在问题。对此，存在多种方案可对其加以处理。一种方法是在出现问题时将旋转设置为 0，当采用该解决方案且球杆穿过球体时，球杆将会出现闪烁现象。针对这一问题，可记录最后一次旋转位置，以使球杆的形状状态保持不变。此时，当鼠标位于母球上时，球杆首先呈现其应有状态且不存在前述旋转行为；随后，可将形状设置为默认值 0。

当按下鼠标键时，可适当修正当前向量。出于简单考量，可将其设置为该方向上的单位变量，该方向直接转换为球体的初始方向。当前，所需工作仅剩确定球体的速度。

当定义球体的运动速度时，可采用前述回拉球杆这一简单方案。也就是说，可记录鼠标键按下时的时间值，随后，在各时间步内，击球的全部时长为已知项。相对于母球，当计算球杆的新位置时，可将该值乘以球杆向量的逆向量（以及一个恒定的缩放系数）。若上述时间值超出了最大允许时间量，则球杆将自动击打母球；否则，当前操作将进入下一个时间步。

当释放鼠标键或超出最大时间值时，可按比例设置母球的速度。其中，一类最为简单的方法是设定最大初始速度。随后，可计算击球时间值与最大时间值的比率，并将该值与最大速度相乘即可得到母球的速度。从视觉效果上看，当移动球杆并击打球体时，该过程类似于再次将球杆即刻置于球体位置处。随后，球杆在隐藏并显示最终结果之前可于此处停留片刻。

11.3.2　游戏主循环

待母球设置完毕后，下面将讨论物理模拟，该过程将相应的数据值代入至前述章节所讨论的各函数中。针对各时间步，下列内容显示了须执行的各项计算：

（1）确定运行时间值，计算相对于最近一个时间步的时间值。

（2）添加摩擦力。针对各移动球体，可通过固定的摩擦数据值降低球体的速度，并以此添加摩擦力。若速度为 0，则对应球体标记为静止状态。

（3）计算潜在碰撞行为。针对各运动球体，分别考察其他球体、库边以及落袋，进而查看潜在碰撞。

（4）若不存在碰撞，则将全部球体设置为新位置并结束当前操作。

（5）计算首次碰撞。在碰撞检测过程中，记录碰撞时间值（作为全部时间的比率值）以及碰撞法线。此类数据值由碰撞检测函数予以返回。

（6）确定碰撞时刻。为了获取碰撞时间值，可将时间比率乘以全部时间值。碰撞时刻与速率、方向向量的乘积结果加至当前位置处，即可将全部球体设置于当前时刻处的具体位置。

（7）处理碰撞。resolveInelasticCollisionFixed()函数负责处理球体和库边之间的碰撞行为。对于两个球体之间的碰撞，则可使用 resolveCollisionEqualMass()函数。据此，可分别设定碰撞球体的最新速度。

（8）重复操作。将全部时间值减去碰撞时间即可获得当前时间步的剩余时间，并返回至步骤（3）。

moveBalls()函数提供了对应示例，进而实现了步骤（1）～（8）。该函数较为冗长，在实际操作过程中，读者可将其划分为多个函数。此处，该函数合并为一个整体以便于读者理解其中的内容。另外，该函数基本等同于 checkCollision()函数，且在若干处实现了优化操作，如下所示：

```
function moveBalls(t, table, cushions, pockets, balls, r, f, e)
  //apply friction and deactivate where appropriate
  set mv to 0
  repeat for each b in balls
      if b is moving then
        set the speed of b to max(the speed of b-f,0)
        if the speed of b is 0 then
          set b to not moving
        otherwise
          set mv to 1 //there is some ball moving
        end if
      end if
  end repeat
  if mv is 0 then return "stopped"

  //check for collisions
  repeat while t>0
    set mn to 2
    //mn is going to be the minimum time proportion
    set ob1 to 0
    set ob2 to 0
    repeat with i=1 to the number of balls
      set b1 to balls[i]
      set pos1 to the position of b1
      set v to t*the velocity of b1
      //its speed * its direction vector
      //check for collisions between balls
      repeat with j=i+1 to the number of balls
        set b2 to balls[j]
        if b1 or b2 is moving then
          set pos2 to the position of b2
          set u to t*the velocity of b2
          if possiblecollision(pos1, v, pos2, u) then
```

```
        set c to
          circleCircleCollision(pos1, v, r, pos2, u, r)
        if c is not a collision then next repeat
        set tm to the time of c
        set m to min(mn,tm)
        if m<mn then
          set mn to m
          set n to the normal of c
          set ob1 to b1
          set ob2 to b2
          set tp to "ball"
        end if
      end if
    end if
  end repeat
  //check for collisions with cushions
  if b1 is moving then
    repeat for each w in table
      //table is an array of two-element arrays,
      //as defined in defineTable()
      set c to circleLineCollision(r, pos1, v, w[1], w[2])
      if c is not a collision then next repeat
      || check if it is minimal just as before;
      || if so, set n, tm and ob1, and set tp to "wall"

    end repeat
    repeat with p in cushions[1]
      //cushions is a two-element array representing the
      //pocket entrances, the first element being
      //a list of circle centers, the second being
      //the radius of the circles.
      if possiblecollision(pos1, v, p, (0,0)) then
        set c to
        circleCircleCollision(pos1, v, r, p, (0,0), cushions[2])
        if c is not a collision then next repeat
        || again, check if minimal, and if so
        || set n, tm and ob1, and tp= "wall"
      end if

    end repeat
  end if
end repeat

if mn=2 then exit repeat //no collision

//otherwise there is a collision

//move balls to collision position
repeat for each b in balls
  if b is moving set its position to
        its position + mn*t*its velocity
```

```
    end repeat
    //resolve collision
    if tp=#wall then
      set u to the direction of ob1
      set the direction of ob1 to
        resolveInelasticCollisionFixed(u, e, e, n)

    otherwise
      set u1 to the velocity of ob1
      set u2 to the velocity of ob2
      set res to resolveCollisionEqualMass(u1, u2, n)
      set the speed of ob1 to magnitude(res[1])
      if this speed>0 then
        set the direction of ob1 to norm(res[1])
        and set ob1 to be moving
      otherwise set ob1 to be not moving
      || repeat for ob2

    end if

    //decrease time and repeat
    set t to t*(1-mn)
    end repeat

    //move balls for the last section (no collisions)
    repeat for each b in balls
      if b is moving set its position to its position + t*its velocity
  end repeat
end function
```

需要说明的是，moveBalls()函数并未判断球体是否落袋，对此，读者可参考练习 11.1 以完成此项任务。

11.3.3　基本的剔除操作

剔除操作是指移除或忽略特定环境集合中的某些无关元素，例如，在 3D 模型中，根据此项操作确定对象的可见（不可见）部分，进而仅绘制对象的可见表面以节省计算时间。

在此类物理模拟环境中，“剔除”意味着碰撞预处理过程，并于随后执行实际测试操作。需要注意的是，在某些场合下，剔除操作并非必需。若剔除操作的执行时间超出了碰撞检测自身，则应对其予以忽略。此时，对应操作仅涉及碰撞圆和直线，此类数据易于检测，当数量较少时尤其如此。实际上，若仅存在少量的碰撞检测计算，则当前模拟过程可得到有效的改善。另外，若涉及较大数量的球体对象，增加剔除操作并不能获得明显的改善。

一类较为常见的剔除技术是将游戏场景划分为多个子空间。例如，假设撞球游戏中的当前桌面状态如图 11.4 所示。其中，桌面可划分为 8 个正方形。球体 A 朝向球体 B、C、D 运动，由于 A 和 B 位于非连续方格内，因而暂时不会发生碰撞。这也意味着，针对球体 A，无须在球体 C

和 B 之间执行碰撞检测。

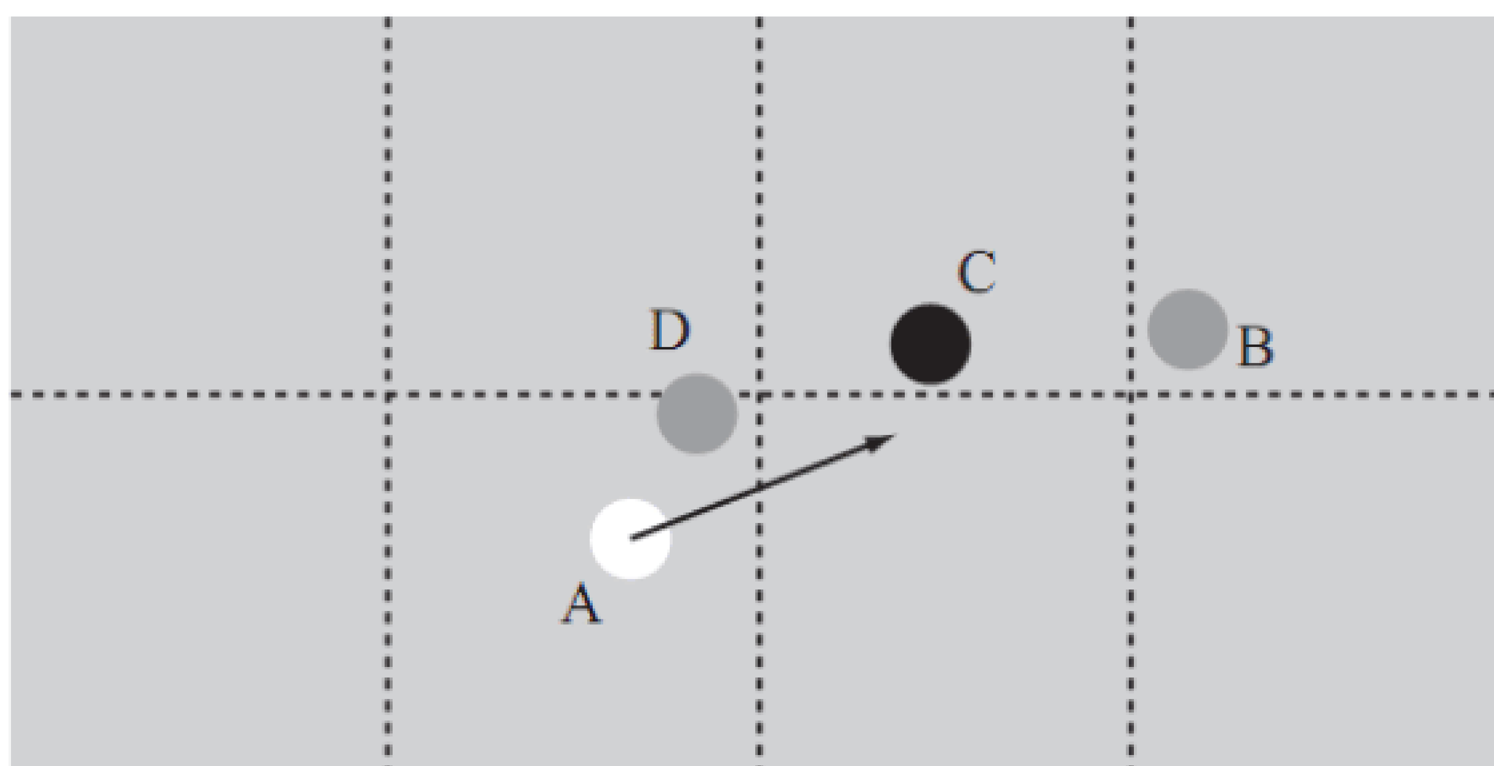


图 11.4　将桌面划分为连续的子空间

另一种划分方案则使用了交叠子空间，如图 11.5 所示。其中，某一球体通常一次性位于多个方格中，这意味着，无论球体速度如何，在球体的全部运动过程中，通常可计算出球体所处的某一子空间。随后，只需检测该空间内的碰撞即可。从本质上讲，该方案等同于首个方案，只是相同的计算被设置在不同的处理部分中而已。针对上述两种处理方案，最终选择结果则视读者个人喜好而定。

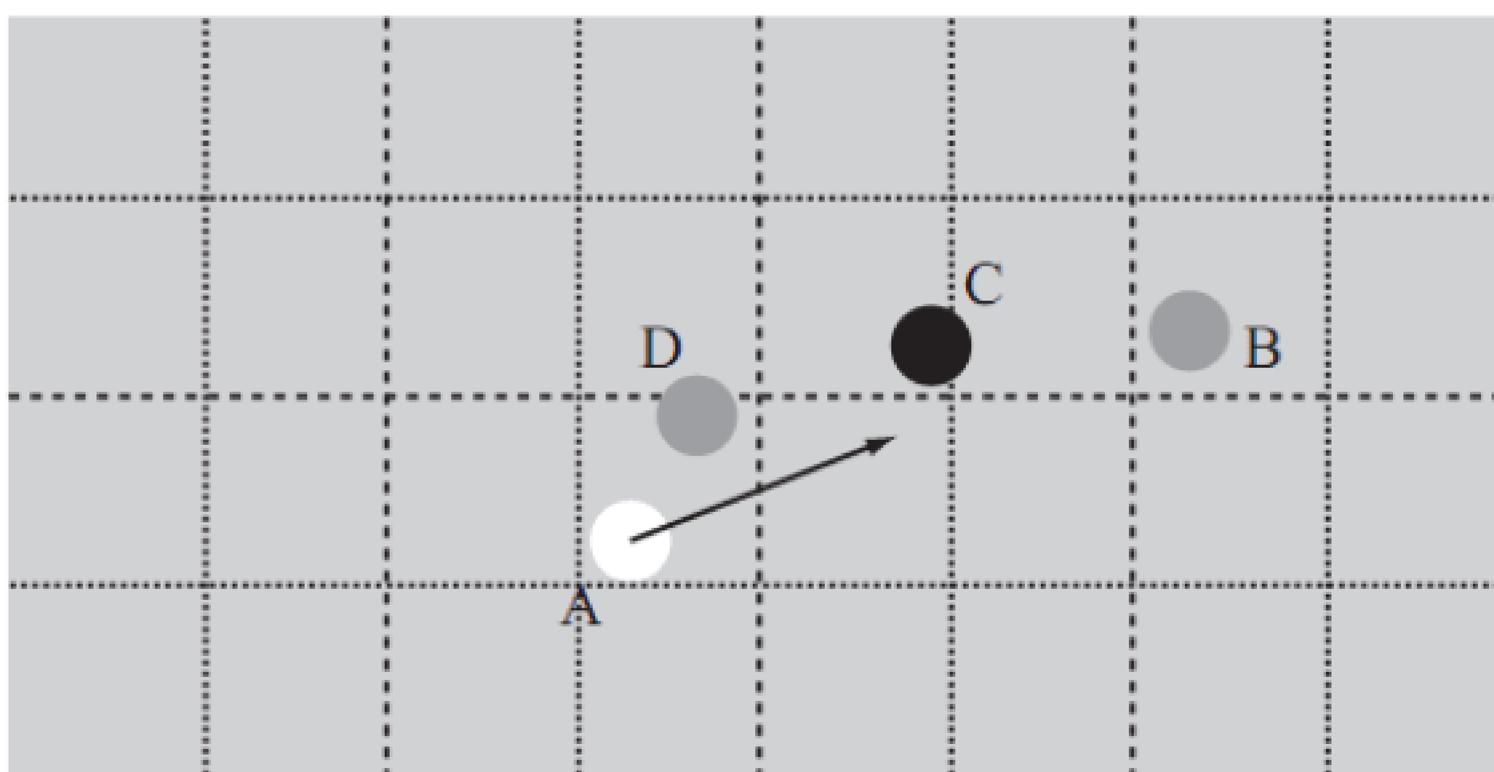


图 11.5　将桌面划分为交叠的子空间

需要注意的是，在上述两种剔除方案中，应确保正方形足够大，以避免球体可一次性地穿越某一方格。对此，可设置最大速度，以及当前模拟环境中的最小时间步。此类措施十分必要，在游戏体验过程中，模拟过程可能因各种原因被中断，从而导致碰撞检测之间的“间隙”无穷大。若最大速度为 m 像素/秒，且正方形尺寸为 s 像素。则最大允许时间步为 $\frac{s}{m}$。实际上，由于球体包含非 0 半径值，因而上述方法并非万无一失，位于某一方格内的球体可与邻接方格内的球体发生碰撞。据此，可将时间步现定于 $\frac{(s-2r)}{m}$，其中，r 表示球体半径。

11.3.4　游戏逻辑

添加游戏逻辑可视为当前游戏中的最后一个阶段，游戏逻辑包括犯规、游戏顺序以及玩家的竞技结果。尽管大多数逻辑元素与数学内容无关，但以下两点内容应引起足够重视，并可将数学

系统转换至当前游戏中：

（1）确定球体是否被正确地击中。这里，技术犯规与一轮击球中首个被击中的球体关系紧密。moveBalls()函数返回首个被击中的球体颜色，进而可用于确定首个被击中的球体。若颜色错误或球体未被击中，则可判为技术犯规。

（2）处理落袋球体。落袋球体将从当前游戏中被移除。moveBalls()函数将对此予以考察，并忽略碰撞检测过程中的落袋球。在每一轮击球结束时，将检测落袋球的颜色并查看是否符合击球规则，以判断当前玩家是否展开第二轮击球。如果落袋球不符合规则，则视为玩家犯规。另外，黑球是否落袋将视为玩家赢得（或输掉）比赛的重要依据。

11.4 本 章 练 习

【练习 11.1】试修改 moveBalls()函数，以判断球体是否落袋。相应地，存在多种方案可对其加以处理，且暂不存在所谓最优方案。

【练习 11.2】试添加“预测”函数并以此判断首先被击中的球体及其移动方向，同时，另输出内容显示于球杆所指之处。

11.5 本 章 小 结

本章讨论了前述函数的组织方式，并实现了一款简单的撞球游戏，本书后续章节还将多次提及该款游戏。在本书第 3 部分中，还将进一步考察更为复杂的物理行为，并将其添加至撞球游戏中，例如摩擦力和旋转。

至此，读者应掌握如下内容：

- 如何将碰撞检测及其处理函数应用于真实的游戏环境中。
- 如何简化物理计算并处理特定的游戏需求。
- 基本的剔除操作，例如空间的划分方式，进而降低计算量。

第 3 部分　复杂运动

本书第 3 部分将继续介绍力学知识，以及某些相对复杂的概念，例如角运动、弹簧、振荡以及运行轨迹。同时，该部分内容还将正式阐述力学知识，其中包括牛顿物理学、简谐运动、摩擦力等话题。随后，本部分还将简要介绍一类复杂的数学曲线，即 Bezier 曲线。

第 12 章　力和牛顿定律

本章包含如下内容：

- 概述。
- 作用力。
- 重力。
- 火箭和卫星。

12.1　概　　述

虽然“作用力”这一术语被多次提及，但截止到目前为止，尚未对其加以准确定义。在这一领域中，物理学研究始于艾萨克·牛顿（1643～1727 年）所开创的运动定律，该定律涉及物体的运动行为以及万有引力作用，进而体现了一类普遍的科学命题。

牛顿经典理论沿袭了数百年，然而，阿尔伯特·爱因斯坦（1879～1955 年）指出了其时空局限性。尽管如此，牛顿动量在日常生活中依然可体现其应有的准确性。

除了作用力定义之外，本章还将对前述内容予以回顾，进而实现数学和物理学之间的有机结合。

12.2　作　用　力

术语“作用力”常出现于人们的日常谈话中，例如论证的说服力，大自然的力量，以及法律的效力。虽然具体含义源自各自的会话环境且往往带有一定的比喻性，但均突出体现了某种功效，且常与“能量”或“作用”等词汇互换使用。在科学领域，作用力则表示为一个技术名词。其中，牛顿通过 3 个定律对其予以准确定义，因而作用力单位冠名为牛顿，即 1 牛顿定义为 $1\text{kg}\cdot\text{m/s}^2$。

12.2.1　牛顿第一定律

最初，人们普遍认为，运动对象需要在推动力作用下方可沿其路径保持移动，否则该对象将会逐渐减速。该理论由亚里士多德（383 B.C.E.～321 B.C.E.）提出且深入人心。然而，该理论并非正确。在日常生活中，该情形随处可见。毕竟，汽车在关闭发动机后处于减速状态。又如，若读者手中持有一辆玩具汽车，在手部停止施加作用力时，玩具将即刻处于停止状态。因此，仅当

处于 0 摩擦力环境下，牛顿理论方得以展现，例如外太空。此时，物体所受的摩擦力极为有限。

对此，牛顿提出了第一运动定律，根据该定律，如果物体未受到外部作用力，则物体将以恒定速度运动。这里，运动行为也包括 0 速运动。也就是说，除非受到力作用，否则对象将保持静止状态。

作用力的形式并未体现于牛顿第一定律中，当前，读者可将其想象为某种影响力。例如，若某一粒子未受到其他粒子的影响，则该粒子将以同一方式持续运动。

如图 12.1 所示，物体未受到任何作用力，因而处于平衡状态。其中，作用于该对象的作用力之和为 0，根据牛顿第一运动定律，处于平衡状态的对象将保持恒定速度。

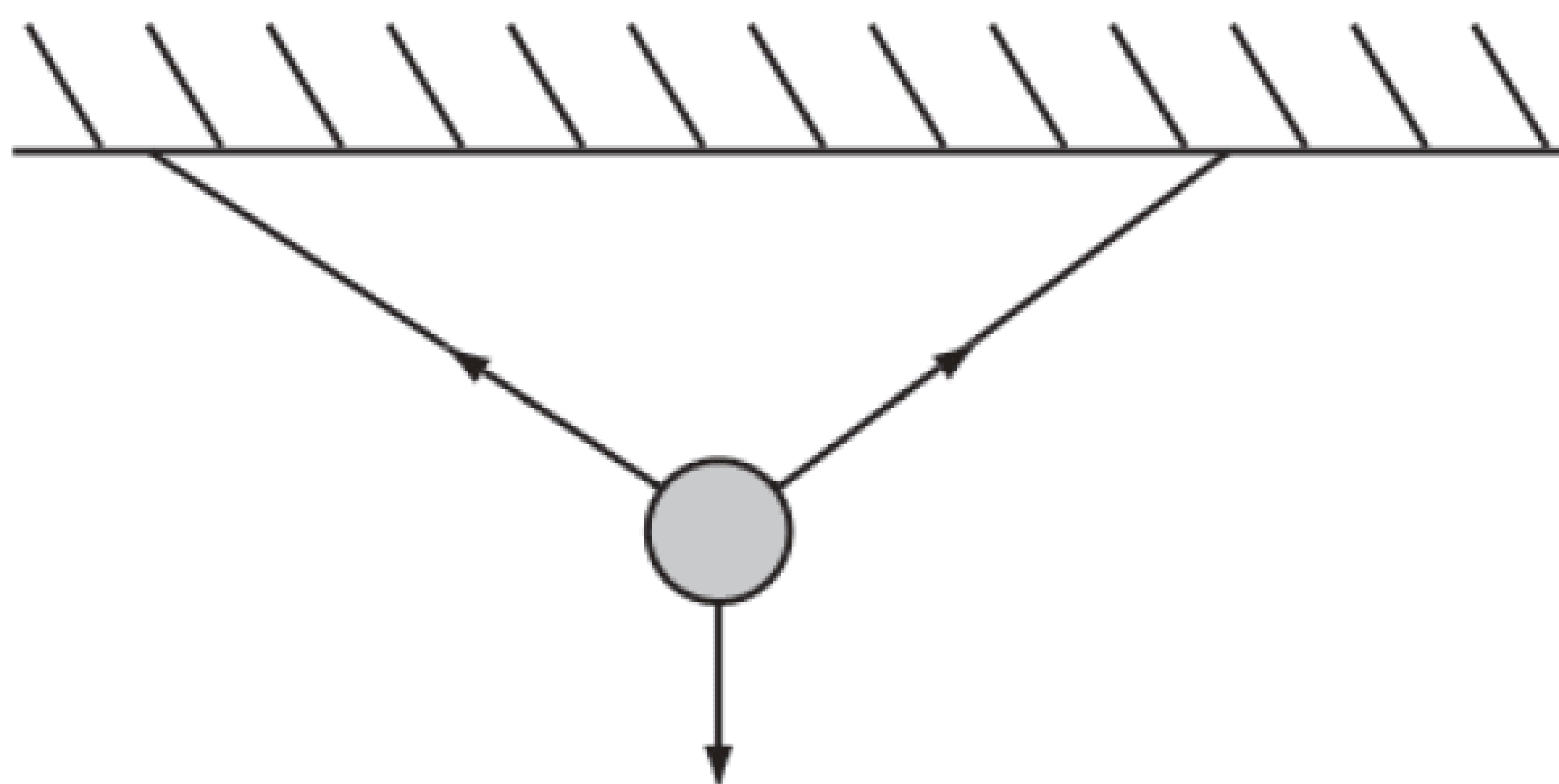


图 12.1　在多个作用力作用下，处于平衡状态的对象

实际上，牛顿第一定律表示为与动量守恒相关的、后续物理定律的早期版本，前述撞球游戏即隐式地使用了这一结论，也就是说，若球体未受到摩擦力或其他球体的撞击，则该球体将以恒定速度运动。

12.2.2　牛顿第二定律

与牛顿第一定律相比，牛顿第二定律则涉及较多的数学计算，其核心内容可描述为：若物体受到外力作用，将处于加速运动状态，并与作用力大小呈正比，且与对象的质量呈反比，其数学形式如下所示：

$$作用力=质量\times加速度$$

关于牛顿第二定律，另一种描述方式则是，作用于物体上的作用力等于物体动量的变化率，当处理作用力或质量时（在一段时间内持续变化），该结论十分有用，例如向心力或火箭的运动状态，本章后续内容将对此加以讨论。

牛顿第二定律定义了作用力的单位，即单位作用力可表示为：质量为 1 千克的物体 1 秒内加速为 1m/s 时所需的作用力。如前所述，该作用力以牛顿命名，其 1 牛顿可简写为 1N。在前述章节中曾讨论到，海平面处的重力加速度约为 $10m/s^2$，这也意味着，海平面处质量为 1 千克的物体，其重力约为 10N。若该作用力作用于读者的身体之上，则读者可感受到自身的体重。

不同于前述章节中所讨论的运动方程，作用力与质量紧密关联。例如，若读者抛 10kg 的炮弹，对应作用力 2 倍于 5kg 的炮弹。此处，作用力与动量相关，而质量则可理解为惯性。质量表

示物体与作用力行为之间的抗拒程度。

12.2.3 牛顿第三定律

如何理解物体的静止状态？根据牛顿第二定律，若对象受到向下恒定作用力，则应处于向下加速状态；否则，必存在一个向上的作用力以使当前物体处于平衡状态。例如，若读者端坐于座位上，则座椅施加作用力以防止读者处于下落状态。相反，座椅也受到了来自读者的向下作用力。相应地，座椅处于静止状态，其原因在于该座椅还受到了源自地板的向上作用力。牛顿第三定律中，若物体受到源自其他物体的外力作用，则前者向后者施加大小相同、方向相反的作用力。

鉴于牛顿第三定律常被错误地使用，因而应对其加以准确描述。作用力包含等值反向的反作用力，但二者并非作用于同一物体上。牛顿第三定律表明，作用力处于对称状态。这里，物体未受到外力，则无法施加作用力。例如，地球通过重力“拉近”物体，与此同时，该物体也在轻微地“拖曳”地球。

鉴于地球表面物体质量与地球质量之间的显著差异，上述行为所产生的效果并不明显——基于物体作用力的地球加速度极其微小。除此之外，地球还受到大量的、来自各方向上的此类微小作用力，因而对应效果彼此抵消。即使是月球这一质量较大的物体也很少令地球偏离其（围绕太阳的）运转轨道。然而，潮汐现象可视为月球引力的最佳证据。

若牛顿第一定律体现了平衡对象的动量守恒，那么，牛顿第三定律则阐述了碰撞对象。实际上，若暂不考虑其他形式的能量（例如热能），该定律可归类于能量守恒定律。牛顿第一定律和第三定律体现了弹道学和碰撞行为的不同视角，而牛顿第二定律则稍有不同，该定律可视为一类数学计算结果，并可用于运动学计算。尽管牛顿第一定律和第三定律在后续章节中时常出现，但牛顿第二定律则更值得关注。

12.2.4 冲量

当处理刚体碰撞时，须重新审视牛顿第二定律。其中，碰撞体速度在碰撞时刻瞬间改变，该过程蕴含了无穷大加速度，因而对应作用力也趋于无穷大。在当前碰撞检测环境下，碰撞结束时的速度则难以计算。

为了有效地解决这一问题，需要引入冲量这一概念。冲量定义为作用力与时间的乘积，牛顿第二定律表明，冲量等于动量的变化。在刚体碰撞中，无穷大的作用力与无穷小的时间值彼此抵消，进而生成冲量。第 9 章曾使用了能量守恒和动量守恒定律，因而可对冲量予以计算。

【提示】 *在真实的碰撞过程中，碰撞体将产生变形且彼此回弹。由于变形和回弹均需经历某一过程，因而可对真实碰撞的时间值进行计算。*

根据冲量的广义定义，读者可尝试重写碰撞计算。当两个对象产生碰撞时，将在碰撞法线 $\mathbf{n}$ 方向上生成冲量 J，这将产生动量变化（为 $\mathbf{n}$ 的倍数），如下所示：

$$m\mathbf{v} = m\mathbf{u} + J\mathbf{n}$$

这里，冲量大小相等且方向彼此相反，也就是说，两个对象分别受到 J 和$-J$ 的作用。动量方程与能量方程经整合后，则可计算 J 值，如下所示：

$$J=\frac{-m_1m_2\mathbf{u}\cdot\mathbf{n}}{(m_1+m_2)}$$

在上述方程中，假设 **n** 为单位向量，**u** 表示为两个碰撞前的相对速度。读者可尝试推导该结果并查看是否与第 9 章中的内容相匹配。

12.3　重　　力

本小结将讨论重力和及其与行星运动之间的作用方式，这一命题始于牛顿。除了牛顿三定律之外，牛顿的另一发现则是重力。其他天文学家则观察并记录了行星运动数据，例如约翰尼斯·开普勒（1571～1630 年）。牛顿从苹果落地这一现象认识到了行星的运动方式，并视为一项革命性的发现。据此，太阳系中的全部行星均向太阳方向处“陨落”且彼此作用。

12.3.1　万有引力定律

作为一类普遍现象，重力作用于全部物体上。与其他基本力（fundamental force）不同，重力只具备引力特征，这与其他作用力形式截然不同。例如，磁力具有吸引力和排斥力。在当前宇宙中，重力仅具备引力这一特征。

重力通过反平方关系施加其作用。两个物体之间的万有引力与二者间的平方距离呈反比关系，此处，“二者间的距离”是指其质心间的距离。万有引力公式如下所示：

$$F=\frac{\mathrm{G}m_1m_2}{d^2}$$

其中，m_1 和 m_2 表示为两个物体的质量，d 表示为二者间的距离，G 表示为常量，即万有引力常数，该值约为 $6.673000\times10^{-11}\ \mathrm{m}^3\ \mathrm{kg}^{-1}\ \mathrm{s}^{-2}$。

既然重力工作于全方位模式下，那么，物体中全部分子间的引力又当如何？针对于球状物体而言，引力效果彼此抵消。另外，大多数此类物体均可视为质心处的粒子。

12.3.2　重力作用下的行星运动

通过考察行星运动，牛顿得出了反平方关系这一结论。作为其研究的基础内容，牛顿借鉴了开普勒定律。经过大量的观察以及细致的工作，开普勒逐渐取代了尼古拉·哥白尼的太阳系学说。哥白尼认为，行星以恒定速率并以圆形轨迹围绕太阳旋转，开普勒则对此持保留意见。

根据开普勒的观点，行星相对于太阳以椭圆方式运动，且太阳位于椭圆的焦点。由于行星轨道的离心率较小，因而开普勒和其他天文学家并未获得精准的运行轨迹。除此之外，开普勒还进一步指出，行星围绕太阳的转动速度并非恒定。实际上，该速度确实处于变化中。在任意时刻，

行星扫掠的椭圆扇形面积保持不变，如图 12.2 所示。当行星靠近太阳一端时，与远端相比，其运行速度明显增加。图 12.2 采取放大方式对此予以说明，对于行星而言，其与圆形轨道的偏差并不明显。

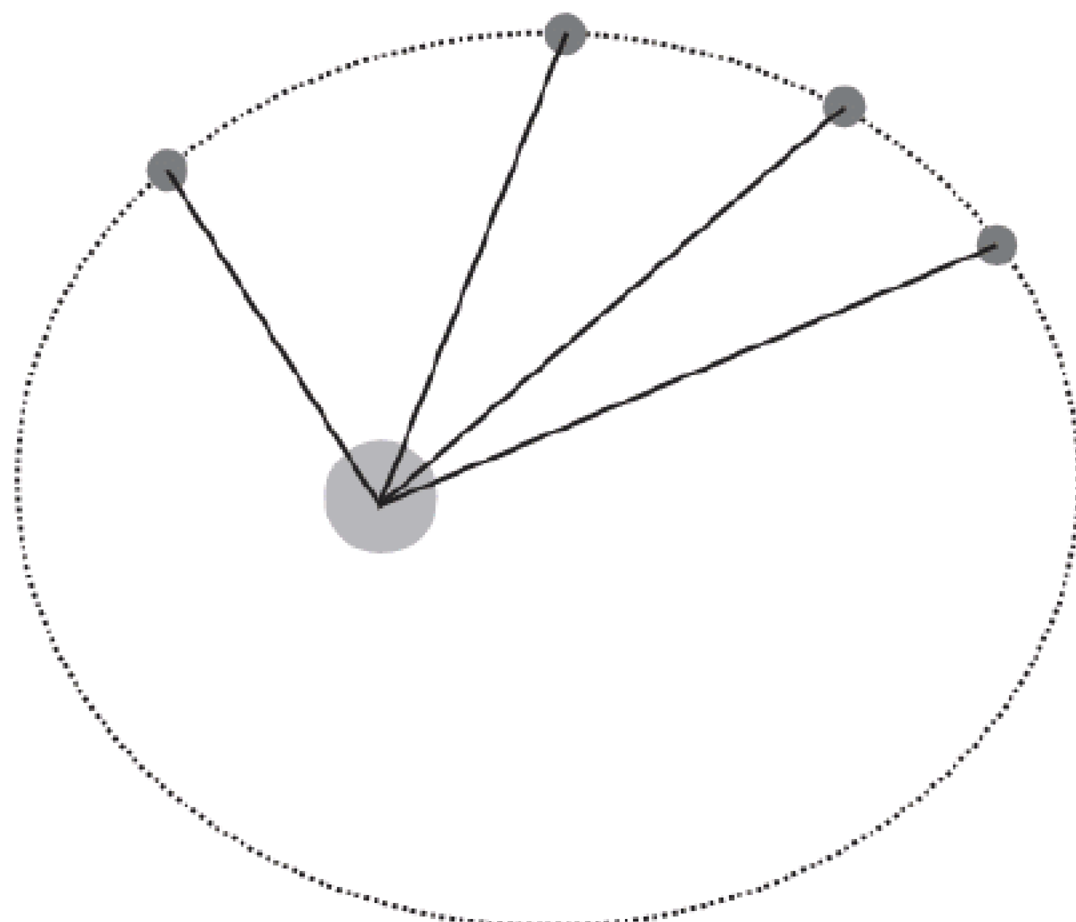


图 12.2　轨道不同处的行星速度

开普勒的观察结果并非仅适用于行星，实际上，相关理论适用于全部轨道天体，例如月球、彗星以及小行星。不仅如此，其使用范围还包括处于脱轨状态并穿越太阳系的陨石。此处，位移差别在于陨石以抛物线方式运动而非椭圆，且等面积定律依然适用。牛顿根据此类观察结果运用了逆向计算方式，并推导出了前述反平方定律。

12.3.3　稳定轨道

全部行星（包括地球）均以螺旋方式缓慢地接近太阳，运行于不稳定轨道的行星通常难以生存，且在几亿年前即已消失。除了地球之外，其他现有行星基本运行于各自稳定的轨道上。需要说明的是，完全稳定的运行轨道并不存在，除了其他方面之外，源自其他行星的引力即可使运行轨道发生偏离，进而使其处于不稳定状态。

关于稳定轨道，较为重要的内容包括运行周期及其半长轴。这里，运行周期记为 T，且定义为一个完整周期所占用的时间。若某一物体的质量远远大于另一个物体，例如围绕太阳运转的某一行星，则上述各数据值通过下列公式彼此关联：

$$T=2\pi\sqrt{\frac{a^3}{\mathrm{G}M}}$$

其中，M 表示为两个物体的全部质量。

周期的倒数称作轨道的角频率，其值等于 2π 倍的角速度，或既定时间内轨迹的数量值，即开普勒第三定律，如下所示：

$$\frac{2\pi}{T}=\sqrt{\frac{\mathrm{G}M}{a^3}}$$

其中，$\frac{2\pi}{T}$ 值也称作平均运动 n，因而初始方程可重写为 $n^2a^3=\mathrm{G}M$ 这一简单形式。

另一个有用的结果可描述为，在距离椭圆中心位置 r 处，可计算粒子的速度，并通过能量守

恒予以求解，进而生成如下算式：

$$v=\sqrt{GM\left(\frac{2}{r}-\frac{1}{a}\right)}=\sqrt{GM\left(\frac{2a-r}{ar}\right)}$$

尽管如此，计算特定时刻 t 的轨道运行体的位置依然难以通过代数或微积分方程求解。

12.3.4　离心力和向心力

圆周运动对象貌似违背了牛顿第一定律，其运动方向不断产生变化，且速度也处于持续变化中。实际上，该过程不存在任何问题。当计算圆周运动时，始终存在一个线性力作用于对象上，并指向圆心位置。该作用力称作向心力，并可对其实施准确计算。若质体 m 以速度 v 实现半径为 r 的圆周运动，则向心力可表示为 $\frac{mv^2}{r}$。若采用角速度 ω 加以描述，则向心力可表示为 $\omega^2 r$，这一关系还将在第 13 章中加以讨论。

这里，读者应区分向心力和离心力之间的关系。例如，基于圆周运动的粒子仅受到内向作用力，根据牛顿第三定律，该粒子还应向保持其旋转状态的其他物体施加作用力。又如，如果读者通过绳索旋转某一物体，则还将存在一个与向心力大小相等、方向相反的外向作用力，该作用力称作离心力。类似地，若转动一个盛满水的水桶，水桶向水施加内向作用力；相反，水也向水桶施加外向作用力。

针对圆周运动，外向作用力的体验源自牛顿第一定律，对应运动趋势是保持直线运动。此处，惯性使得体验者感觉受到了外向力的作用。对此，假设搭建飞车走壁表演中所使用的一组死亡之墙，并在墙壁内部表演趋于圆周轨迹的运动，此时墙壁将施加指向圆心的作用力，以使表演者保持其应有位置。若表演者位于墙壁外侧，则定会以切线方式飞离轨道，而非径直向外。

12.4　火箭和卫星

在第 7 章讨论弹道学时，曾假设重力作用呈现为恒定加速度效果，在当前阶段来看，这一结论并不正确，且重力随高度而变化。相对于地球，若物体尺寸较小，则这一差别并不明显；当处理空间物体时，则须对可变重力予以关注。

12.4.1　地球静止轨道

1945 年（十几年后，首颗人造卫星发射成功），科幻小说作家 C. Clarke 发现，静止轨道的运行周期随距离而变化，在距地球表面某处，轨道卫星的运行周期为一天。这也意味着，若卫星轨道与赤道平行，该卫星将在地球表面上方保持同一位置。如果电信技术中使用了卫星网络，那么，Clark 的推测并非空穴来风。其中，静止轨道卫星视为通信、检测以及 GPS 技术的核心内容。

读者可尝试计算地球静止轨道卫星的正确高度，需要注意的是，针对轨道中运行的物体，都

需承受与重力大小相等的向心力，对应公式如下所示：

$$\frac{mv^2}{r}=\frac{mMG}{r^2}$$

若卫星的运行周期为 T，则有 $v=\frac{2\pi r}{T}$，因此

$$\frac{4\pi^2 r}{T^2}=\frac{MG}{r^2}$$

$$r=\sqrt[3]{\frac{MGT^2}{4\pi^2}}$$

在上述方程中，可带入与地球相关的各项数据值，其中，地球的质量为 6×10^{24}kg，其恒星日（自转周期）为 86164s。这里，恒星日是指地球相对于恒星完成一周旋转所需的时间。地球的恒星日稍短于太阳日，后者是指相对于太阳完成一周旋转所需的时间。由于地球围绕太阳旋转，因而该过程涵盖了额外的相对旋转。最后，r 可取值为 42168km。

当前科幻小说还描述了太空仓这一设备，该设备由光纤连接静止轨道卫星和地面点。当太空舱就位后，这将极大地降低空间所需的能源消耗。当然，随之而来的技术问题也不容小视。首先，光纤的重量巨大，且与航天器的接驳过程也存在一定的危险性。其次，尽管卫星运行于静止轨道，但连接光缆并非如此，因而电缆有可能使得卫星脱离运行轨道。尽管如此，大量的科学团队从未停止其研发的脚步。

12.4.2 高速飞行的炮弹

截止到目前为止，前述讨论尚未提供与火箭运行轨迹相关的处理工具。其中，一类较为关键的问题是，在燃料消耗过程中，火箭的质量不断降低。关于质量处于变化状态的对象，后续章节将依次对其进行讨论。在开始阶段，可考察高速飞行中的炮弹对象，此类对象以较高的速度在空中飞行，其质量保持恒定，下面将介绍其行为方式。

首先，炮弹的质量远远小于地球，因而可忽略炮弹对地球的引力作用。另外，此处假设炮弹在海平面处向空中垂直发射。

对此，可从能量角度对炮弹发射后的运动方式予以考察。其中，在距地心 x 位置处，炮弹的势能表示为 $\frac{GMm}{x}$。若炮弹的初始速度为 u，则运动过程中有 $\frac{1}{2}mu^2=\frac{1}{2}mv^2+\frac{GMm}{x}$，对应微分方程如下所示：

$$\left(\frac{dx}{dt}\right)^2=\frac{2GM}{x}-u^2$$

某些时候，微分方程难以通过代数方式进行求解，此处也不例外。然而，若给定初始位置和炮弹的发射速度，则可通过逐步增量法计算一段时间内炮弹的运行状态。moveCannonBall()函数即根据这一方式予以编写，如下所示：

```
function moveCannonBall(currentHeight,initialSpeed,timePeriod,G, M)
  set currentSpeed to sqrt(2*G*M / currentHeight- initialSpeed * initialSpeed)
```

```
  return currentSpeed * timePeriod
end function
```

12.5　本 章 练 习

【练习 12.1】试编写一组函数，以使行星系在重力模式下运动。

尽管存在多种方案可实现这一任务，但该任务所涉及的数学内容基本相同。在各个时间步内，可根据其他行星的引力场计算各行星的受力状态，并于随后将其转换为线性加速度。需要注意的是，应根据时间步初始阶段的位置计算加速度（不使用移动后的位置数据）。最后，判断是否可“构造”一个行星以使其可在轨道上平滑运行。

12.6　本 章 小 结

本章以全新方式回顾了早期的科研成果，读者应深入理解其中的内容，进而在程序设计中熟练应用力学知识和牛顿定律。第 13 章将继续讨论天体运行知识，并重点介绍角运动和角动量。

至此，读者应掌握如下内容：

- 牛顿三定律及其与能量和动量的关联方式。
- 万有引力定律以及如何计算行星的运动方式。
- 地球静止轨道的计算方式。
- 炮弹对象的发射过程及其运动行为的计算方式。

第 13 章　角　运　动

本章包含如下内容：

- 概述。
- 杠杆物理。
- 旋转。
- 旋转碰撞。
- 向撞球游戏中加入旋转行为。

13.1　概　　述

前述章节探讨了对象的线性运动，但该运动并非是唯一的运动方式。相比较而言，旋转陀螺并不包含线动量，但依然是一类较为常见的运动方式。本章将重点考察角运动，或自旋物理，并将该运动行为添加至第 11 章展示的撞球游戏中。

13.2　杠 杆 物 理

与角运动相关的话题多源自杠杆，杠杆是人类发明的一种简单机制，并将杆状对象置于某一支点上。通过调整支点的位置，可较为方便地支撑重物。与抬动手提箱相比，通过杠杆，稍加力气即可支撑重达数百磅的重物，杠杆的威力可见一斑。作为古希腊最负盛名的数学家之一，阿基米德（280 B.C.E.～211 B.C.E.）曾对杠杆进行了系统的研究。关于杠杆的功效，他曾描写道："给我一个支点和一根足够长的杠杆，我就能撬动整个地球"。

在大多数与杠杆相关的讨论中，并未考虑杠杆和支点的强度。对此，假设二者的强度可支撑起任何重物。当然，实际情况并非如此，在实际应用中，必须考虑到材质的强度问题。具体而言，缺乏足够强度的杠杆将产生弯曲现象，而支点则有可能破损或坍塌。出于讨论目的，忽略材质的强度将大大简化杠杆的计算过程。本章采用轻质杠杆这一概念描述质量为 0，且不会弯曲、折断的杠杆。如未作特殊说明，后续计算均会采用此类杠杆。

13.2.1　转矩

如图 13.1 所示，当作用力施加于杠杆某处时，将导致杠杆处于旋转状态。杠杆左侧的向下

作用力 F 使杠杆加速向下运动，支点则施加向上的反作用力–F。由于这两个作用力位于杠杆的不同处，因而部分杠杆向下运动，而杠杆的另一部分则向上运动，这一形状作用力称作转矩。

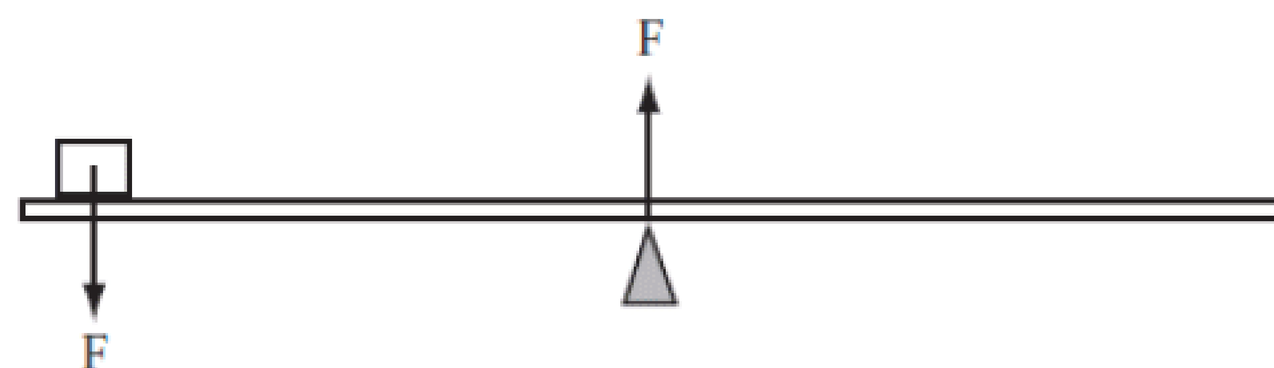


图 13.1　作用于杠杆上的转矩

为了进一步理解转矩的计算方式，首先查看作用力 F，且该作用力偏离支点一段距离。同时，支点仅在接触点处施加作用力，因而其他位置处的作用力将导致旋转运动。实际上，旋转量取决于作用力与支点之间的距离，转矩公式可描述为：

转矩=垂直作用力×距支点的距离

除此之外，上述公式还可采用下列方式加以定义：

转矩=作用力×距支点的垂直距离

关于转矩，上述两种描述方式的差别如图 13.2 所示。实际上，二者彼此等价（证明过程涉及两个公式的点积计算）。

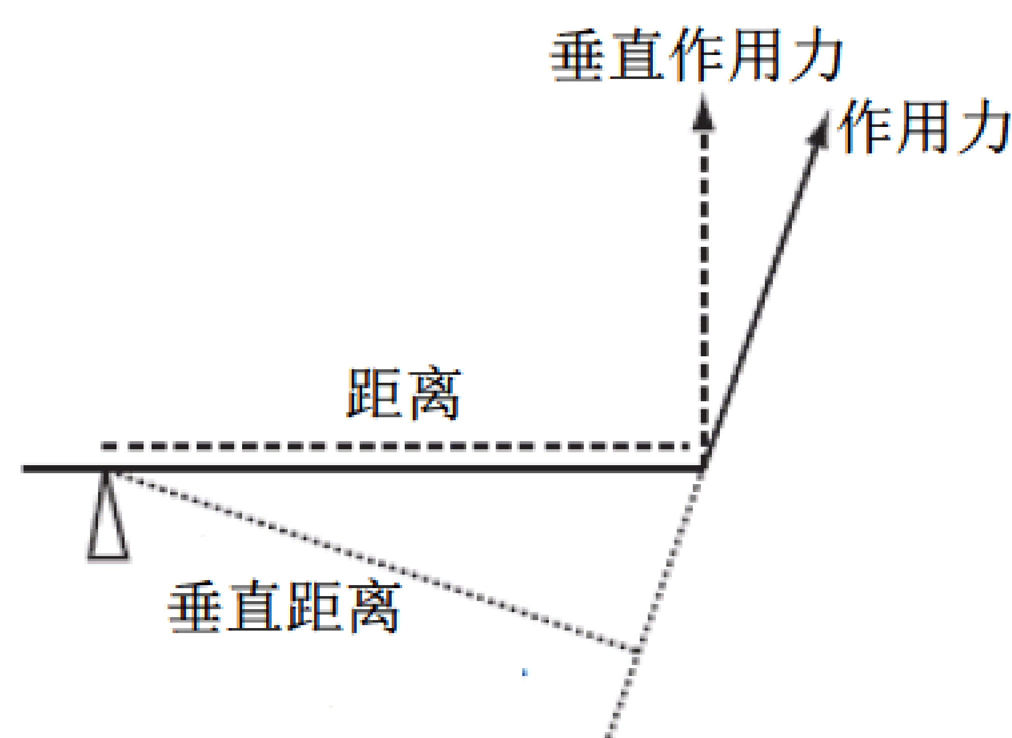


图 13.2　转矩计算

在图 13.2 中，支点与作用点时间的距离表示为向量值。若作用力施加于支点的另一侧，则转矩也将逆置。最终，读者可将该操作视为杠杆，并可通过平衡于某一支点处的平面加以描述。随后，作用力可应用于平面上的任意一点。对应转矩可表示为支点距作用力位置之间的向量函数。作用力大小与距离之间的乘积也称作力矩，通常，转矩即表示为作用力的力矩。类似地，相关计算还包括速度矩或动量矩。

若两个作用力同时作用于杠杆上，则对应转矩可彼此叠加。如果二者之和为 0，则杠杆处于平衡状态。这也意味着，可通过远离支点的较小作用力平衡支点近处的较大作用力。据此，可计算置于杠杆上的重物的重量。

当计算物体的重量时，假设其重量为 W，如图 13.3 所示。此处，已知该物体距支点的距离为 x。当设置重量为 A 且平衡距离为 y，则有如下公式：

$$Wx-Ay=0$$

$$W=\frac{Ay}{x}$$

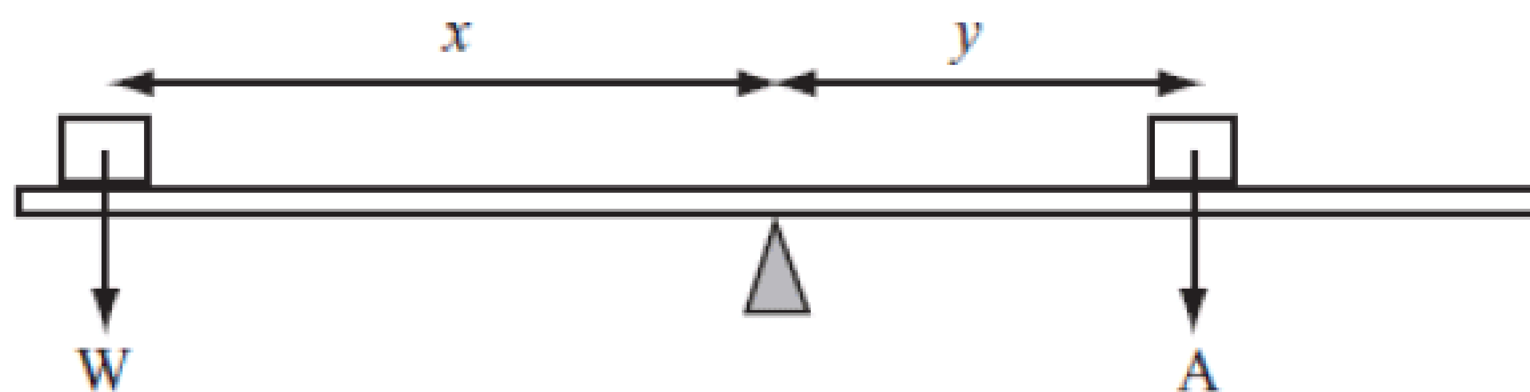


图 13.3　作用于杠杆上的两个作用力

需要注意的是，由于杠杆上的物体皆处于静止状态，因而可得出如下结论：对象的作用力均处于平衡状态。也就是说，存在一个大小为 W 的作用力并向上施加于第一个物体上，该作用力可视为其他对象所施加的转矩结果。这一关系可予以进一步明确说明，即针对任意静止于杠杆的对象，其垂直作用力大小等于杠杆上其他对象的转矩之和。随后，可将该值除以对象距支点的距离。了解了这一点，读者也就理解了杠杆与古代投石机的工作原理。

特别地，假设杠杆静止于地面上，其上球体距支点为 x 且重量为 W，此时，若在杠杆另一侧距支点 y 处放置重量为 A 的物体，则球体受到基于其他对象的作用力。这里，假设杠杆接近于水平状态，作用力约为 $\frac{Ay}{x}$ 并垂直于当前杠杆。

最终，球体以垂直于杠杆的方向加速运动，与此同时，杠杆自身也发生旋转。作用力持续作用，直至球体的飞行速度大于杠杆的旋转速度，抑或杠杆的另一端击中地面而停止运动。此时，球体脱离杠杆。

总体而言，从 Ay 和 x 之间的比值可明显地看出，随着重量于支点处的切线分量不断减小，转矩也将随之降低。除此之外，其他因素也可影响杠杆的行为，例如杠杆的重力及其潜在的断裂可能性。此处，杠杆的重量并不是问题，可将其视为一个附加转矩，并可通过如图 13.3 所示的方法进行计算。对于杠杆的潜在断裂，其计算过程稍显复杂，断裂现象取决于杠杆材质的强度，即剪切力或弯矩。除非模拟过程需要精准的计算结果，否则该问题可予以忽略。

13.2.2　转动惯量

旋转物理与线性运动具有许多相似之处，在旋转行为中，牛顿定律、运动方程、动量以及能量均包含对应项。例如，可计算物体的角速度或角加速度。针对全部旋转量值，需要确定旋转中心以及旋转轴，且仅可包含一个旋转轴。若物体围绕轴 A 旋转，且不包含线性运动，则围绕其他轴向的角速度为 0。

在旋转操作中，与质量对应的数据项称作转动惯量，前述杠杆示例中对此已有所提及。这里，术语“力矩”的用法稍有不同，并包含了平方距离因子。相应地，杠杆的转动惯量定义为相对于支点的、基于平方距离的全部质体的乘积之和。在图 13.4 中，杠杆围绕支点的转动惯量为 $3\times4 + 4\times1=16\text{kg}\cdot\text{m}^2$。

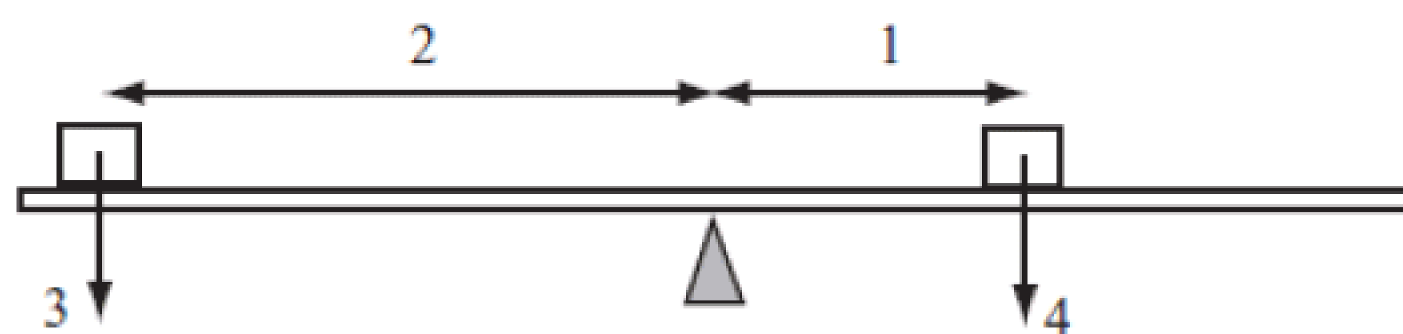


图 13.4　计算转动惯量

当处理真实物体而非轻质杠杆时，转动惯量的计算变得越发复杂，其原因在于，须通过积分运算对无穷多个微小片段执行求和计算。例如，读者可尝试计算质量为 m、长度为 21 的均匀杠杆相对于其中心位置的转动惯量。由于质量均匀分布，因而长度为 δ 的杠杆片段其质量为 $\dfrac{m\delta}{2l}$。转动惯量的积分运算如下所示：

$$I=\int_{-l}^{l}x^2\left(\frac{m}{2l}\right)dx$$
$$=\left(\frac{m}{2l}\right)\left[\frac{x^3}{3}\right]_{-l}^{l}$$
$$=\frac{ml^2}{3}$$

又如圆形对象，读者可将其视为无穷多个同心圆构成的序列。由于圆环均匀分布，且各圆环距圆心相同距离。围绕垂直于圆形且穿越圆心的轴向，各圆环的转动惯量为 $m_x x^2$，其中，m_x 表示为半径为 x 的圆环的质量。

这里，令圆环的宽度值为 δ，且假设 δ 值无穷小。在（距原点的）半径 x 处，对于质量为 m 且半径为 r 的圆形，圆环的质量约为 $\dfrac{2\pi x\delta m}{\pi r^2}=\dfrac{2x\delta m}{r^2}$。若针对全部圆环执行积分运算，则可得到下列算式：

$$I=\int_0^r x^2\times\frac{xm}{r^2}dx=\frac{m}{r^2}\left(\frac{x4}{4}\right)_0^r=\frac{mr^2}{4}$$

对于某些读者来讲积分的具体含义可能难于理解，然而，当前的重点是如何使用此类公式，而非其推导过程，读者只须关注围绕物体质心的转动惯量即可。此处，质心的含义是指，通过该点分割物体，则分割线的两侧具有相同的质量（即质量也一分为二）。

类似的积分运算也可用于计算质心。另外，读者还可计算围绕任意轴的转动惯量，该过程并不复杂。若轴向 A 穿越质心 I，且对象的质量为 m，则距 A 相距 p 处的平行轴其转动惯量为 $I+mp^2$。

13.2.3 惯性片状物体

片状物体可视为另一类对象，并可描述为无限薄的平面对象，例如盘状物体或正方形，图 13.5 显示了对应的转动惯量。首先，薄平面内包含两个平行轴，分别穿越该对象上的同一点 O。据此，穿越点 O 垂直于薄平面的轴向，其转动惯量可表示为：围绕其他两轴的转动惯量之和。

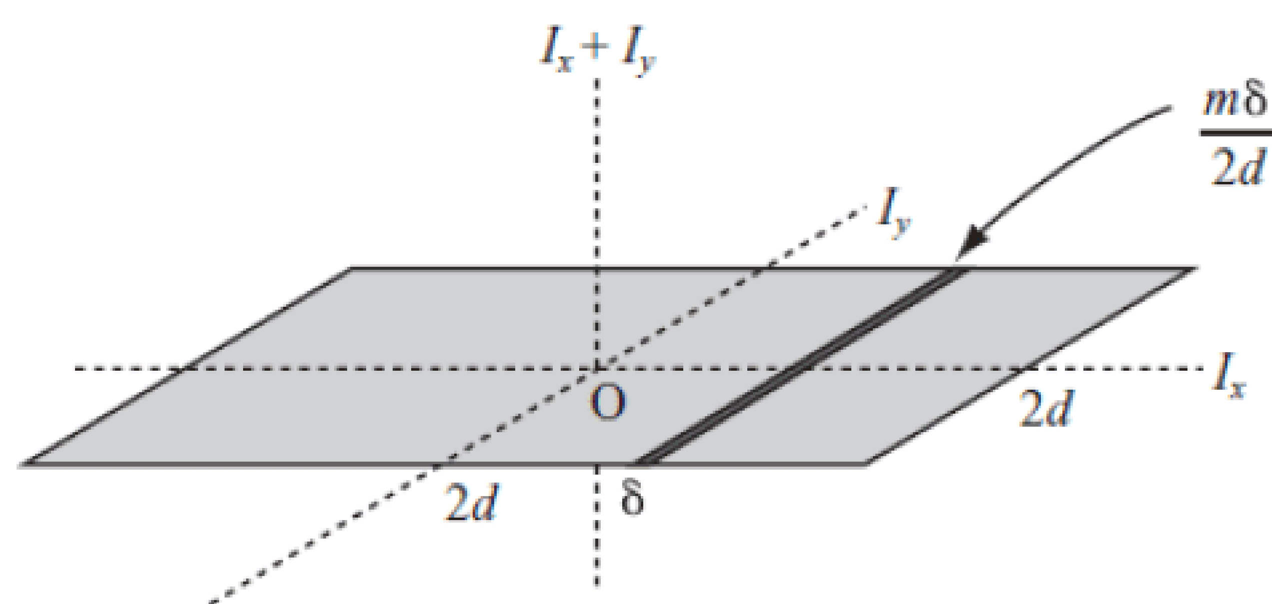


图 13.5 正方形片状对象的转动惯量

根据上述信息，读者可尝试计算边长为 $2d$ 的正方形的转动惯量，该正方形围绕穿越中心位置且垂直于正方形的轴向旋转。这里，假设正方形位于原点位置，相关数据类似于前述杠杆示例，该正方形围绕 y 轴的转动惯量为 $\frac{md^2}{3}$。考虑到对称性，围绕 x 轴的转动惯量基本大同小异。最终，围绕垂直轴的转动惯量可表示为 $\frac{2md^2}{3}$。需要说明的是，片状对象的计算过程也适用于其他简单形状。

如前所述，质量体现了移动某一物体的难易程度，转动惯量同样体现了对象旋转的难度。例如，若物体的质量集中于轴向附近，则该物体具有较小的转动惯量且易于旋转。相反，若物体的质量远离轴向，则该物体具有较大的转动惯量且难以旋转，或者说，此类物体更易于停止旋转。关于惯性，读者可采用牛顿第二定律的变化版本建立下列关系：

转矩=转动惯量×角加速度

13.3 旋　　转

本小节将进一步讨论旋转问题，特别地，通过转动惯量这一概念，读者可尝试计算基于旋转对象或滚动对象的角运动，例如机械装置中的飞轮。

13.3.1 芭蕾舞演员和旋转陀螺

在二维环境下，对象的角动量可定义为角速度与其转动惯量之间的乘积。在三维环境下，角动量的定义则更加复杂，并需要通过张量这一概念予以定义。类似于线动量，角动量也体现了停止某一运动对象时的难易程度。具有较大转动惯量的对象通常包含较大的角动量，因而需要较大的转矩以使其停止。

针对具有较大转动惯量的物体，往往需要较大的转矩促其停止，这也解释了机械装置中的飞轮结构。飞轮可视为一类较大的重型轮状结构，其转动惯量通常较大且须花费较长时间方可令其停止。其中，遏制飞轮运动的主要作用力为源自转轴的摩擦力。由于该作用力接近于旋转轴，对应转矩较小，因而可将飞轮视为一种较好的能量存储方式。

除了旋转轴之外，读者还可尝试计算围绕其他轴的运动物体的角动量。对此，可将对象视为粒子。空间内线性运动的粒子依然具有围绕任意轴向的角动量，即线动量的力矩。如图 13.6 所示，可通过 $m|\mathbf{v}|d$ 这一关系描述该值，其中，d 表示为粒子运动直线与当前轴向之间的垂直距离。

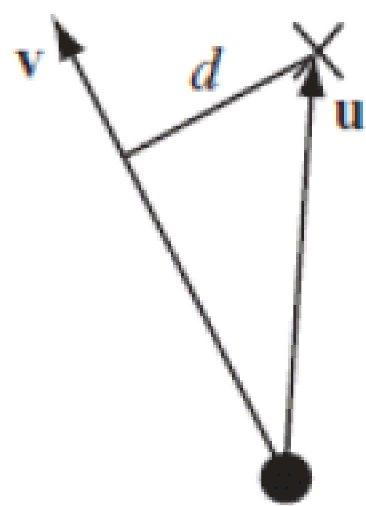


图 13.6　线性粒子的角动量

针对图 13.6 所示的计算关系，须考察计算值为正值或负值，也就是说，有必要了解粒子围绕当前轴向呈顺时针或逆时针方向运动。一种方案是定义顺时针向量法线。随后，可计算顺时针法线 **v** 和粒子距轴线的向量 **u** 之间的点积值，即|**v**|*d*。当且仅当粒子围绕当前轴线呈顺时针运动时，该值为正值。moment()函数即采用了这一方案并返回角动量，如下所示：

```
function moment (position, vector, axisPosition)
  set n to clockwiseNormal(vector)
  return dotProduct(n, axisPosition-position)
end function
```

除此之外，还可使用角速度计算（距中心位置的）向量 **r** 处的点速度，该值等于 $\omega\mathbf{r}N$，其中，$\mathbf{r}N$ 表示 **r** 的顺时针法线。

类似于线动量，角动量同样遵循守恒定律。亦即，若对象的转动惯量发生变化，其角速度也将产生变化并以此作为补偿，这一现象常出现于花样滑冰运动员或芭蕾舞演员身上。当开始旋转时，运动员往往在水平方向上张开手臂；当运动员增加其旋转角速度时，则上举或下垂其手臂，并以此降低围绕垂直轴的转动惯量。

三维对象通常会围绕具有最小转动惯量的轴向旋转，该轴向通常为对称轴。若不存在其他反抗力，对象通常会沿旋转方向移动并朝向当前对称轴，这一现象称作陀螺效应。顾名思义，这在陀螺的旋转过程中较为常见。初始阶段，对象的旋转较为缓慢，一旦与对称轴保持一致，则转速突然增加。旋转变化可视为角动量“滞留”结果，在线性运动中，对象通常并不会自发地加速，而在角运动中，对应结果堪称奇特。陀螺效应常用于枪械中，子弹在枪膛中呈现为回旋运动，由于子弹趋向于与旋转轴保持一致，因而可获得更加稳定的旋转效果。

13.3.2　旋转动能

旋转对象的能量与线性动能类似，旋转动能等于$\frac{1}{2}$×转动惯量×角速度^2。通过观察可知，该表达式的单位等同于常规的动能单位，即 kg • m/s^2。需要注意的是，当前公式中的角速度须采用弧度单位予以标识。

由于旋转动能仅是多种能量形式中的一种，因而可转换为线性动能或重力势能，例如悠悠球。其中，球体在下落时开始旋转。通过控制降落速率，即动量，悠悠球可展现有趣的运动行为。当悠悠球接近地面时，将在返回手中之前开始原地旋转。这里，最高点处的重力势能转换为动能；另外，在悠悠球下降过程中，势能还将转换为旋转动能。在悠悠球降至最低处时，动能和原势能全部转换为旋转动能。随后，旋转动能将再次转换为动能，以使悠悠球获得“攀爬”能力。

需要注意的是，悠悠球的降落速度通常小于在空气中下降的一般球体。由于部分势能转换为旋转动能，因而全部动能及其对应的线速度必然小于非旋转对象。类似地，若忽略摩擦力作用，沿斜面滚动的水泥管将慢于冰块——相同尺寸的实心圆柱体（具有较大的转动惯量）的运动速度相对缓慢。这也从另一个角度证明了伽利略观点的错误之处，即全部物体均以相同速度下降。若物体自由滚动，则较大物体的降落速度将更为缓慢。

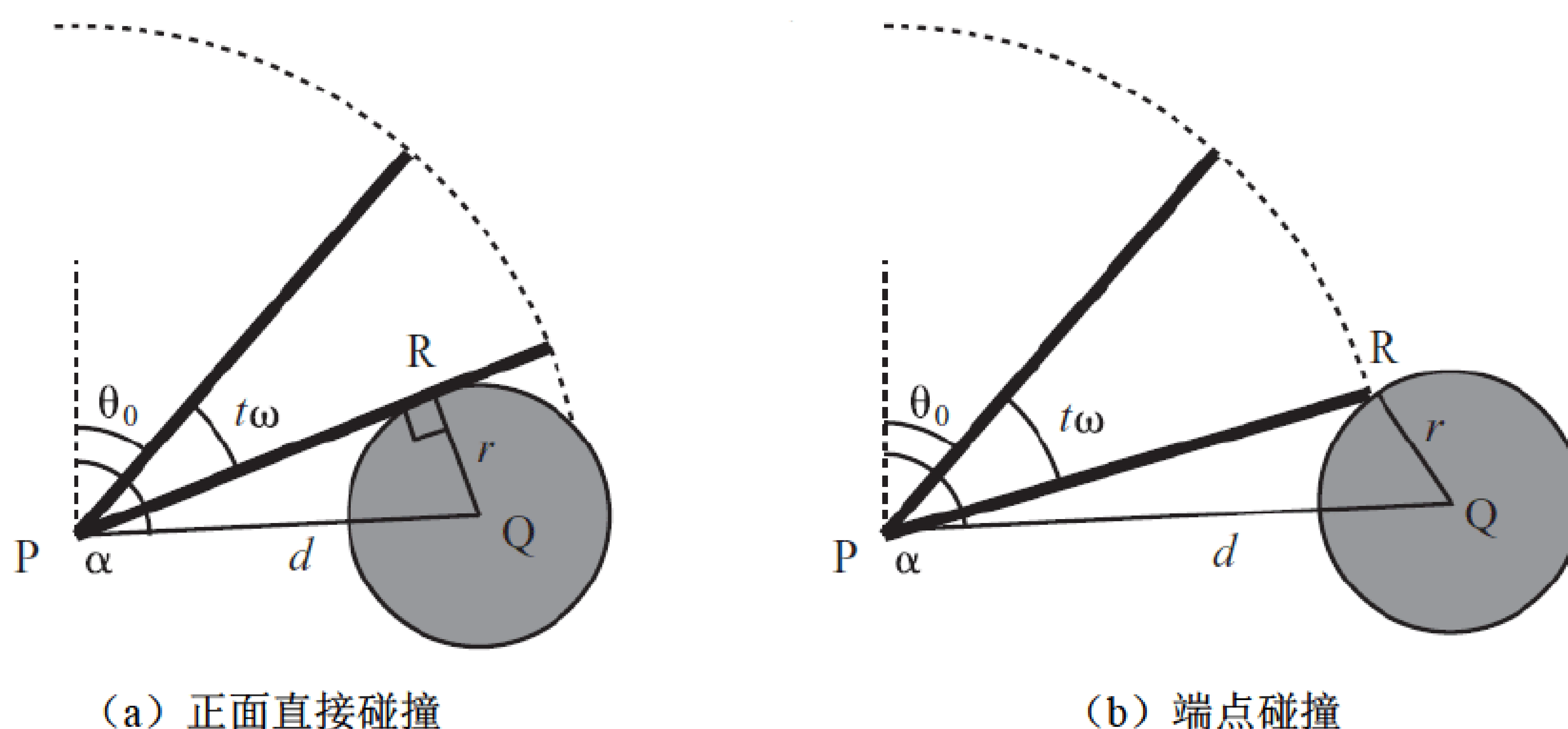
13.4 旋 转 碰 撞

若对象处于旋转状态，则碰撞行为将更为复杂，且需要考察多个因素。其中，以某一角度引发碰撞的非旋转对象，可导致对象间处于旋转状态。另外，旋转对象可能从侧面或前缘边产生碰撞，角动量将对碰撞结果产生显著影响。

正如大多数复杂碰撞类型那样，角碰撞通常难以采用代数方式进行计算，因而常采用近似处理方案。本小节并不打算深入讨论各种可能性，相关示例仅提供了问题的切入点。

13.4.1 旋转直线和圆形之间的碰撞检测

作为第一个旋转碰撞示例，图 13.7 显示了围绕点 P 旋转的直线段，其角速度为 ω，距离垂直方向上的初始角度为 θ_0。另有一圆其半径为 r，圆心位于点 Q，且与 P 之间的距离为 d，与垂直方向间的角度为 α。



（a）正面直接碰撞

（b）端点碰撞

图 13.7　旋转直线和圆

在图 13.7（a）中，假设圆位于直线段范围内，且无须考察直线段的端点。在任意时刻 t，直线段的角度值表示为 $\theta = \theta_0 + t\omega$。另外，当直线段在碰撞点 R 处与圆相切时，二者间构成了直角三角形 PRQ。其中，QR=r 且斜边为 d。若直线段以顺时针方向旋转，则有∠RPQ =$\alpha-\theta$。若直线段以逆时针方式旋转，则∠RPQ=$\theta-\alpha$。这将生成下列方程进而可确定碰撞点：

$$\alpha-\theta=k\sin^{-1}\left(\frac{r}{d}\right)$$

$$\theta_0+t\omega=\alpha-k\sin^{-1}\left(\frac{r}{d}\right)$$

$$t=\frac{1}{\omega}\left(\alpha-\theta_0-k\sin^{-1}\left(\frac{r}{d}\right)\right)$$

针对顺时针运动，k=1；对于逆时针运动，k=-1。

需要注意的是，角度计算通常在一个闭合域中进行。其中，$\alpha+\beta$ 不一定大于 α 或 β。一类最为简单的计算转换方式则是确保数值不超过 360°，或不低于 0°。

如前所述，上述算式仅适用于圆形与 P 足够近时，进而确保其与直线的平直部分接触，而非直线段的端点。若直线段的长度为 l，当且仅当 $d^2 \leqslant l^2 + r^2$ 时，碰撞产生于直线部分。相反，若圆形较远，则二者间不会产生任何接触，即 $d > l + r$。

对于中间范围，情况又当如何？图 13.7（b）显示了这一问题的不同之处。由于圆并未与直线段相切，因而 PRQ 不再是直角三角形。然而，长度 PR 为已知项，即直线段 l 的长度，这也意味着，可通过余弦定理计算 RPQ 角度值，如下所示：

$$\cos(\alpha-\theta)=\frac{l^2+d^2-r^2}{2ld}$$

$$\theta=\alpha-k\cos^{-1}\left(\frac{l^2+d^2-r^2}{2ld}\right)$$

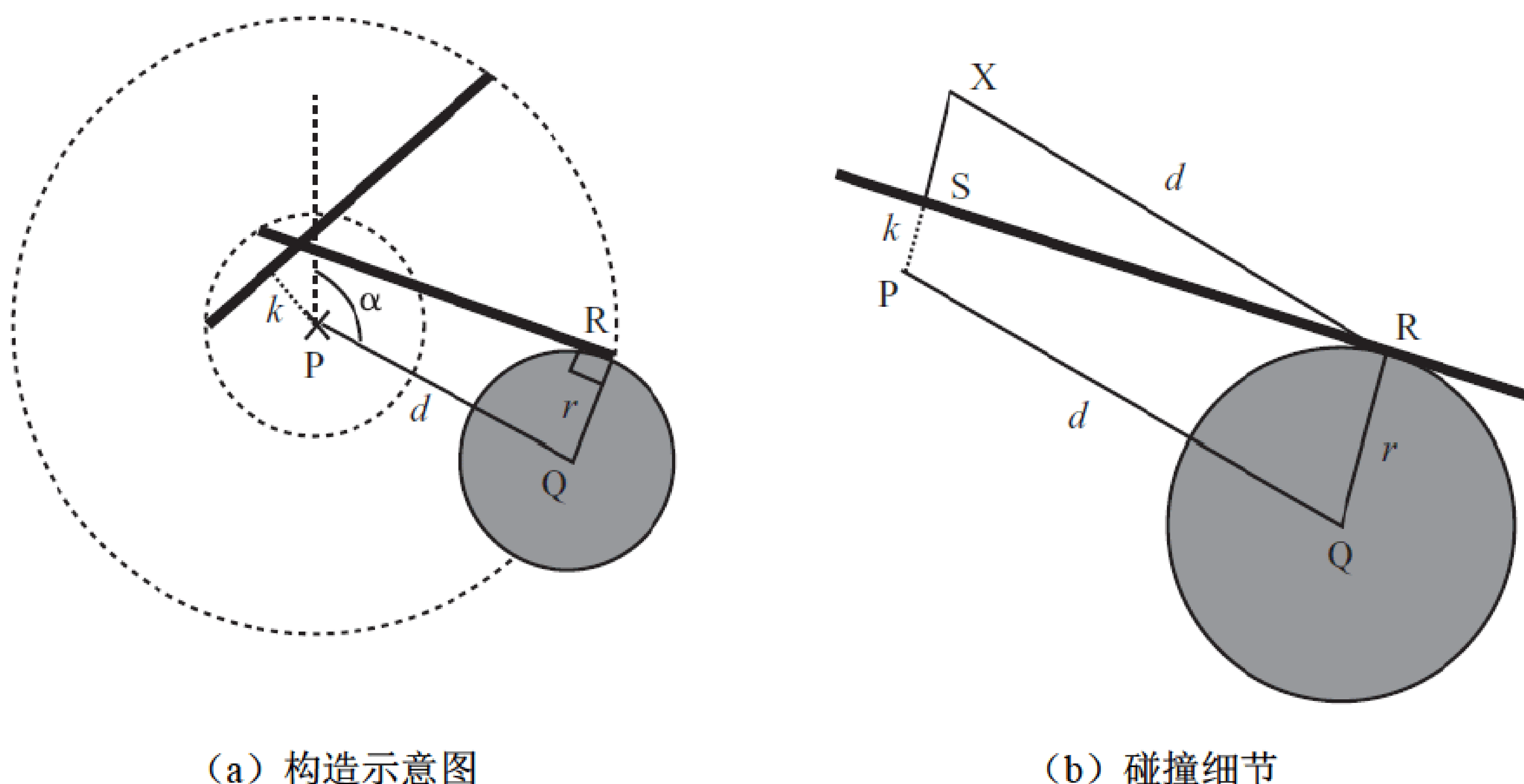
$$t=\frac{1}{\omega}\left(\alpha-\theta_0-k\cos^{-1}\left(\frac{l^2+d^2-r^2}{2ld}\right)\right)$$

据此，可编写 angularCollisionLineCircle()函数，该函数确定旋转直线和圆之间的全部可能碰撞，如下所示：

```
function angularCollisionLineCircle(thet0, omega, l, r, d, alph)
  if d>l+r then return "no collision"
  if d<r then return "embedded"
  //move into a calculation within the range [0,2pi]
  subtract thet0 from alph
  if omega<0 then
    set omega to -omega
    set alph to -alph
    set k to -1
  otherwise
    set k to 1
  end if
  while alph<0 add 2*pi to alph
  while alph>2*pi subtract 2*pi from alph
  //check if there is a possible collision
  if alph>omega then return "no collision"
  //now perform the appropriate collision check
  if d*d<=l*l+r*r then
    return (alph-k*asin(r/d))/omega
  otherwise
    return (alph-k*acos((l*l+d*d-r*r)/(2*l*d)))/omega
  end if
end function
```

类似的计算也适用于旋转点与圆之间的碰撞，例如玩家利用脚趾或靴子踢动足球时。相应地，另一种方式则是将当前问题视为旋转多边形的顶点。

如图 13.8 所示，当直线段围绕某一点（该点并未位于当前直线上）旋转时，正如矩形边那样，实际计算过程并不复杂。通过观察可知，此时直线段围绕点 P 旋转，且与当前直线之间的垂直距离为 k。如前所述，圆位于点 Q 处，且与 P 之间的距离为 d（对应角度值为 α）。



（a）构造示意图　　（b）碰撞细节

图 13.8　围绕偏离点的旋转直线

图 13.8（b）显示了碰撞力矩。其中，R 为圆与直线之间的交点，点 S 表示为垂线（源自点 P）与当前直线的交点。类似地，图中还绘制了直线 PQ。需要注意的是，PS 和 QR 均垂直于当前直线，因而二者处于平行状态。如果从 R 处绘制一条直线并平行于 QP，则该直线与延长线 SP 交于点 X，且与 S 之间的距离为 $r-x$，这将生成直角三角形 XSR。在该三角形中，一条边等于 $r-k$，斜边为 d。据此，可计算角度值 RXS = P – QPS 以及距离 RS。若角度有所偏差，则可通过微调方式对正面碰撞计算予以修正，如下所示：

$$t=\frac{1}{\omega}\left(\alpha-\theta_0-k\sin^{-1}\left(\frac{r-x}{d}\right)\right)$$

由于端点围绕 P 以圆形方式运动（距离为 $\sqrt{l^2+x^2}$ ），因而当前方案与前述处理手法有所不同。考虑到任意旋转多边形可视为围绕某一轴向旋转的直线段集合，所以这一特定的碰撞处理方法十分重要。

13.4.2　圆和运动直线

该问题关注运动圆与直线之间的碰撞行为，与前述内容相比，其处理情形较为复杂。如图 13.9 所示，圆形沿路径 $\mathbf{q}+t\mathbf{v}$ 运动，直线围绕点 P 旋转。在某一时刻 t，圆形与直线彼此接触，图中绘制了两条较为重要的辅助线。

图 13.9 假设直线在角度 0 处开始旋转，为了计算其他情形，首先需要围绕 P 旋转参考坐标系。除此之外，此类计算还假设碰撞现象位于第一个运动四分位中，也就是说，在某一时间步中，直线的角位移小于 90°。当处理较大的角度值时，则需要将当前计算划分为多个子问题。最后，此处假设点 P 位于点(0,0)处。

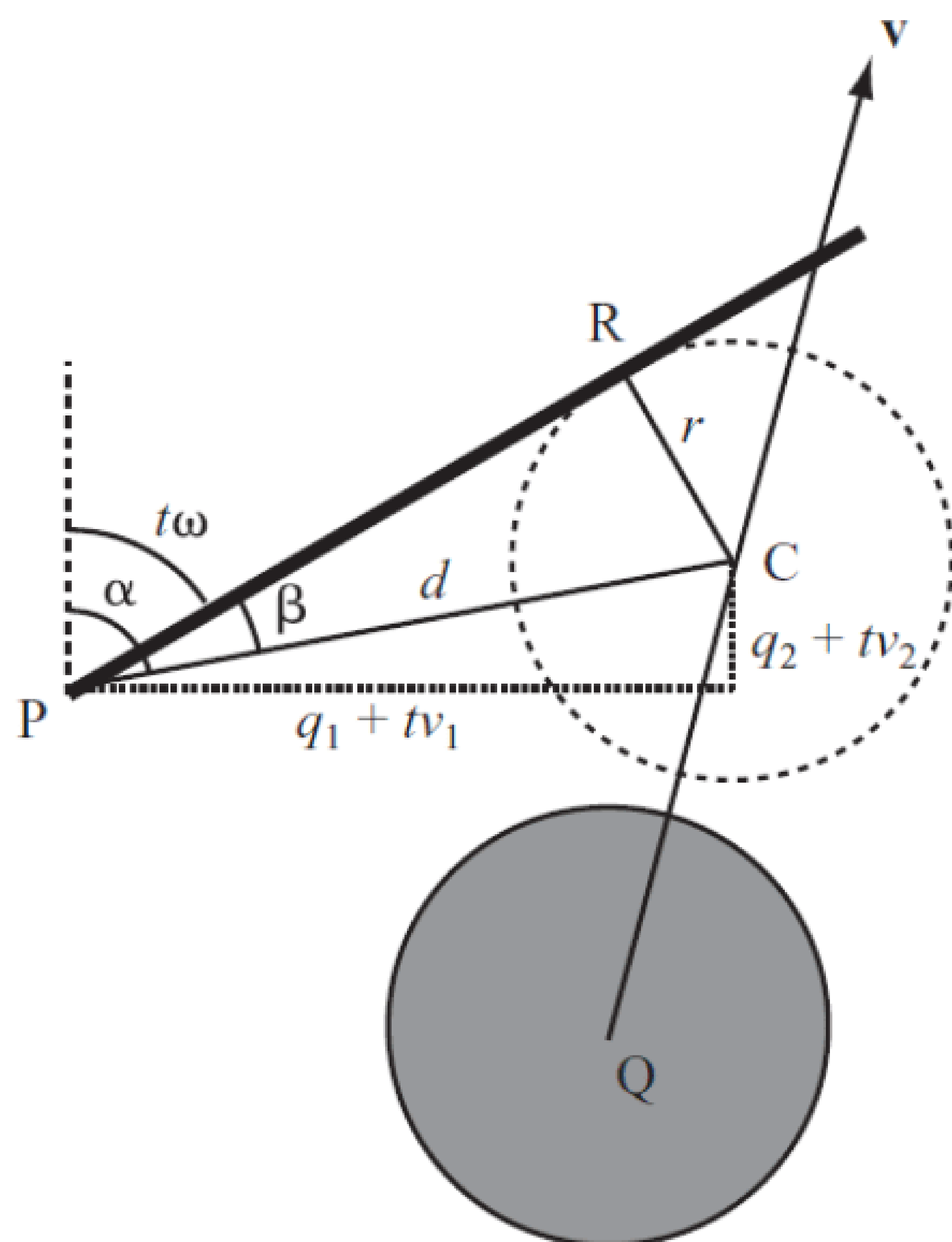


图 13.9　运动圆和旋转直线

为了求解图 13.9 中的问题，首先需要考察 α 和 β 值。这里，α 值表示直线 CP 于碰撞点处形成的角度。其中，C 表示圆心。β 值则表示 CP 和旋转直线于碰撞点处形成的角度。据此，相关结论可描述为：$\alpha-\beta=t\omega$ 或 $\alpha+\beta=t\omega$。通过比较初始位置和两个对象的运动状态，可对上述结论予以进一步的区分。对此，可令 $\alpha=t\omega+k\beta$，其中 $k=\pm1$。

通过三角恒等式以及 d 值，可将 CP 的长度值定义为 t 的函数，方程的目标则是求解 t 值，如下所示：

$$\frac{q_1+v_1}{d}=\sin\alpha$$

$$=\sin(k\beta+t\omega)$$

$$=\cos\beta\sin(t\omega)+k\sin\beta\cos(t\omega)$$

$$\frac{q_2+v_2}{d}=\cos(k\beta+t\omega)$$

$$=\cos\beta\cos(t\omega)-k\sin\beta\sin(t\omega)$$

$$\cos(t\omega)\left(\frac{q_1+v_1}{d}\right)-\sin(t\omega)\left(\frac{q_2+v_2}{d}\right)=k\sin\beta(\cos^2(t\omega)+\sin^2(t\omega))$$

$$=k\sin\beta=\frac{kr}{d}$$

$$\cos(t\omega)(q_1+tv_1)-\sin(t\omega)\ (q_2+tv_2)=kr$$

从前述计算中可以看出，对应方程并未对 t 值予以分离，这可通过代数方式对其进行计算。鉴于方程无法实现精确求解，因而须使用第 6 章介绍的近似方案。这里，由于函数连续，且当前任务是计算基于 0、1 边界的较小的数值范围，因而一类相对安全的方法是采用二分法或 Newton-Raphson 法，进而可方便地计算数据范围外的目标根植。

该函数的实现过程需要检测直线的端点碰撞，其中，基本的代数计算大同小异。此处需保留 d 值，并根据余弦定理获得 β 值，对应方程为 $\cos\beta = \frac{l^2 + d^2 - r^2}{2ld}$。待求解完毕后，读者还将面临更为复杂的方程，如下所示：

$$\sin(t\omega)(q_1 + tv_1) + \cos(t\omega)(q_2 + tv_2) = \frac{l^2 + d^2 - r^2}{2l}$$
$$= \frac{(q_1 + tv_1)^2 + (q_2 + tv_2)^2 + l^2 - r^2}{2l}$$

如前所述，鉴于该方程无法通过代数方式进行求解，因而需采用近似方案。这里，一种节省时间的方法是执行前期检测，进而判断两个对象是否碰撞。实际上，此处应执行两项检测：首先，需要确定当前圆形是否与直线扫掠后的完整圆相交，并于随后检测旋转碰撞。其次，还需进一步确定圆形扫掠的角度是否与直线扫掠后的角度交叠，该检测结果将生成 k 值。

【提示】此处并未考察旋转直线偏离其旋转点这一情形，尽管所涉及的代数方案大同小异，但具体计算过程将十分复杂。

在全部问题中，需要单独处理旋转直线的两个端点，这涉及圆形与静态旋转中心点之间的碰撞检测。否则，读者须执行两次计算，并针对当前直线的反方向，向初始角度值加入 π 值。

13.4.3 直线间的碰撞检测

两条直线之间的碰撞行为看似直观，但依然需要借助于近似处理方案，并通过渐进方式逼近计算结果。图 13.10 显示了旋转直线和静态直线段之间的相交方式，且存在两种情形，分别涉及直线段部分的碰撞，以及某一端点的碰撞。

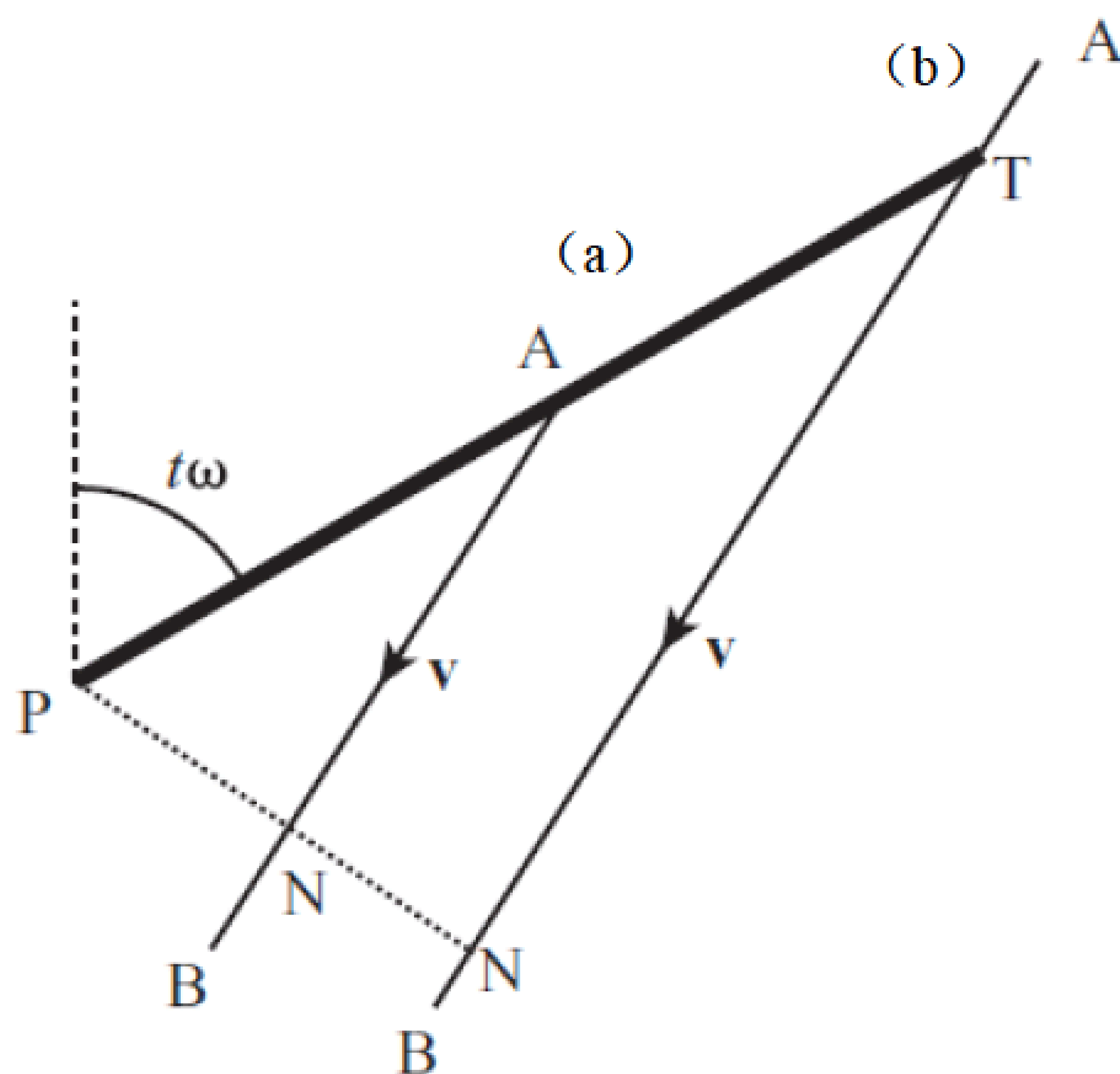


图 13.10　旋转直线和旋转直线段。(a) 端点碰撞；(b) 线段部分的碰撞

当处理此类问题时，一种常见的方法是假设端点 A 的位置向量 **a**，以及该点距另一端点 B 的向量 **v** 已知。除此之外，还可进一步确定点 N，其中，源自点 P（定义为原点）的垂线与当前

直线段相交。此后，处理过程与本章前述方法基本相同。相应地，读者可定义一个较为有用的函数 clockwise()，若源自 **a** 的向量 **v** 围绕原点呈顺时针状态，则该函数返回+1；当逆时针时，则该函数返回−1。在某些场合，该函数还可返回 0 值，并以此表明向量指向或背向原点（当前函数并未涵盖这一功能）。clockwise()函数的实现过程如下所示：

```
function clockwise(p, v)
  set n to clockwiseNormal(p)
  if component(v,n)>0 then return 1
  return -1
end function
```

其中，clockwiseNormal()函数如下所示：

```
function clockwiseNormal(v)
  return vector(-v[2],v[1])
end function
```

clockwiseNormal()函数将顺时针定义为特定方向。当然，若可保证一致性，通常可任意设定该方向且不会产生问题。针对某些显示功能，顺时针方向貌似呈现为逆时针效果。例如，y 值可通过向上或向下方式予以标识。同样，还应确保正角位移的正确使用，对此，可编写两个通用功能的函数，即 unitVector()函数和 angleOf()函数，如下所示：

```
function unitVector(ang)
  return vector(sin(ang),cos(ang))
end function

function angleOf(v)
  return atan(v[1],v[2])
end
```

结合上述两个函数，即可解决相关数据值的一致性问题。

【提示】在三维环境中，上述过程可通过更加正规的方式予以处理，即使用叉积和右手法则，第 16 章将对此加以讨论。

下面返回当前的核心问题，即运动和静止直线间的碰撞检测。这里，首先处理一类简单的情形，即旋转直线与线段交于 A 和 B 之间的点 T。该过程形成了一个直角三角形 PNT，其中，PT 表示为直线的长度，并可生成对应的角度值。另外，若直线以顺时针方式旋转，则会沿 **v** 的逆时针方向碰撞，反之亦然。随后，还需进一步检测 T 是否位于当前直线上，即 $0 \leqslant \overline{AT} \cdot \mathbf{v} \leqslant |\mathbf{v}|^2$ 是否为真。

处理端点则相对简单，若直线的旋转方向与 **v** 相同，则需要查看与 A 之间的碰撞结果。相应地，若直线以其他方式旋转，则需要检测与 B 之间的碰撞结果。对于前者，直线构成了一个直角三角形 PNA，如图 13.11 所示。此处可先期确定 AN 的长度，这将生成正确的角度值。需要注意的是，应确保以正确的方向计算角度值，并检测 N 是否位于 AB 上。换而言之，须判断向量 $\overline{AN}$ 与 $\overline{AB}$ 具有相同的方向。若是，则需要计算逆时针角度，否则，则应计算顺时针角度。

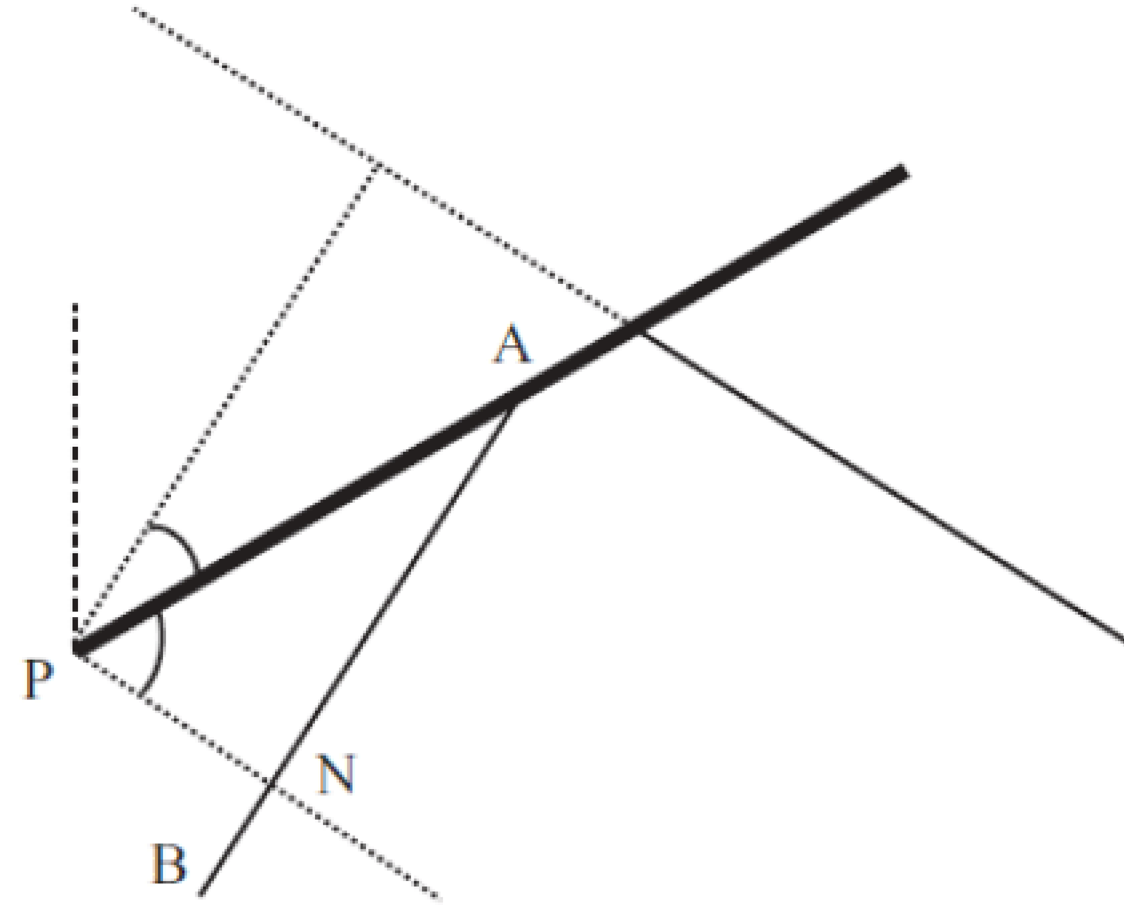


图 13.11　确定碰撞角度与端点之间的正确方向

至此，可编写 angCollLineStatLine()函数，顾名思义，该函数处理静态直线和运动直线之间的碰撞行为，如下所示：

```
function angCollLineStatLine(thet0, angvel, length,linept, linevect, segment)
  //segment=1 if you are checking for end points,
  //0 for a continuous wall
  set n to norm(normal(linevect))
  set d to dotprod(linept,n)
  if d<0 then
    set d to -d
    set n to -n
  end if
  //so n is the normal vector directed toward N
  if d>length then return "none" //too far from wall

  //if checking for end points, see if they are relevant
  if segment=1 then
    set pn to n*d //the vector PN
    set dd to length*length-d*d //the squared length TD
    if angvel>0 then
      if clockwise(linept, linevect)=-1 then
        set endpt to linept
      otherwise
        set endpt to linept+linevect
      end if
    otherwise
      if clockwise(linept, linevect)=-1 then
        set endpt to linept+linevect
      otherwise
        set endpt to linept
    end if
  end if

  set d1 to sqmag(endpt-pn) //sqmag is the squared magnitude
  if d1<dd then //there is a potential collision with the end point
    set a to acos(d/mag(endpt))*clockwise(endpt,pn-endpt)
    //a is the angle of collision with the end point
```

```
    otherwise
      set a to acos(d/l)
      if angvel>0 then set a to -a
      //check if this collision occurs outside the line segment
      set ap to pn+abs(a)*sqrt(dd)/a
      //note that abs(a)/a is 1 if a>0, -1 otherwise
      set k to mag(ap-linept)/mag(linevect)
      if k>1 or k<0 then return "none"
    end if
    otherwise
      //check for collision with an infinite wall
      set a to acos(d/l)
      if angvel>0 then set a to -a

    end if
    set tn to angleof(n)
    set t to rangeangle(tn-thet0+a,1)/angvel
    if t<=0 or t>1 then return "none"
    return t
  end
```

13.4.4　两条旋转直线

系统地讲解旋转直线之间的碰撞行为则超出了本书的讨论范围，但其中仍涵盖了某些本质问题，如图 13.12 所示。由于全部碰撞结果（除了退化现象）均可描述为两个端点之间的准确碰撞，即各次碰撞位于某一直线端点和另一直线（平直部分）之间，因而处理过程相对简单。

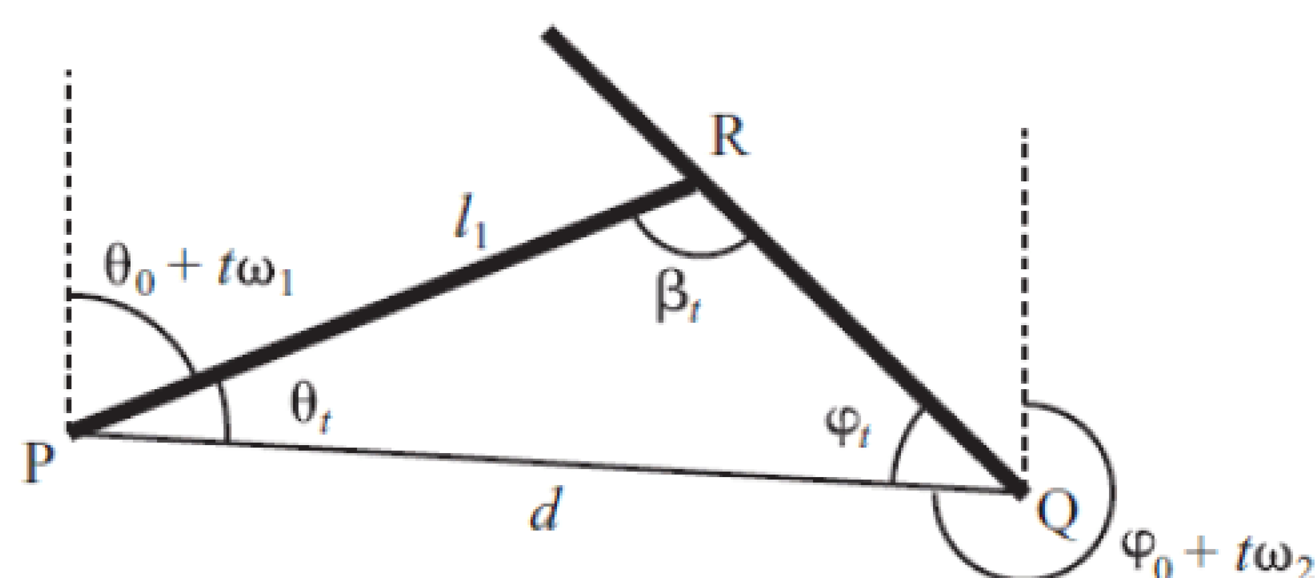


图 13.12　两条旋转直线之间的碰撞

对此，需在前期阶段完成某些预备工作。首先，需要计算直线的角度 α，该直线连接两个旋转点 P 和 Q。根据该角度值，即可定义两个函数，进而计算三角形（该三角形由两条直线和直线 PQ 构成）中的两个角度，如下所示：

$$\theta_t = \alpha - \theta_0 - t\omega_1$$

$$\varphi_t = \varphi_0 + t\omega_2 - \alpha - \pi$$

在三角形 PQR 中，第三个角度为 $\beta_t = \pi - \theta_t - \varphi_t$。为了检测直线间是否碰撞，首先需要确定 β_t 值是否位于 0 和 π 之间，这将生成基于 t 的不等式，如下所示：

$$\pi \leqslant \varphi_0 - \theta_0 + t(\omega_2 - \omega_1) \leqslant 2\pi$$

不难发现，α 项不复存在。若该不等式针对 0～1 之间的任意 t 值均成立，则可在当前时间范围内获取潜在的碰撞三角形。

若该三角形正确无误，则可通过正弦定理确定是否产生碰撞，这需要计算 PR = l_1 且 QR = l_2，或者 PR = l_1。最终将得到下列算式：

$$\frac{\sin\beta_t}{l_1}=\frac{\sin\varphi_t}{d} \text{ 或 } \frac{\sin\beta_t}{l_2}=\frac{\sin\varphi_t}{d}$$

其中，d 表示为 PQ 的长度。

当前，需再次返回至近似方案。需要注意的是，当前计算并未涉及两条直线的运动方向，方向数据将会对三角形 PQR 中所使用的数据值产生显著影响。

旋转中心点之间是否彼此相对运动并不会对当前方法产生影响。然而，读者仍需要适当调整 t 函数中的 d 和 α 值，尽管此类变化使得计算过程趋于复杂化，但对应理论保持不变。

13.4.5 处理角碰撞

与线性碰撞相比，角碰撞的处理难度相对适中。当计算此类碰撞行为时，读者可采用第 12 章引入的理论，并集中考察接触点处的碰撞结果。若两个对象处于旋转状态，则边上一点的速度可能与对象整体速度之间存在一定的差异。回忆一下，在前述计算中，当围绕某一固定轴旋转时，点速度为 $\omega\mathbf{r}_N$。从广义上讲，若对象以速度 $\mathbf{v}$ 运动，则点速度表示为 $\mathbf{v}+\omega\mathbf{r}_N$。

在碰撞时刻，两个对象之间的冲量 J 将影响二者的线动量和角动量。若对象的初始速度分别为 $\mathbf{u}_1$ 和 $\mathbf{u}_2$，则二者的最终速度为 $\mathbf{v}_1$ 和 $\mathbf{v}_2$。相应地，若对象的初始角速度为 ω_1 和 ω_2，则最终角速度为 φ_1 和 φ_2。另外，可采用 m 表示对象的质量，I 表示转动惯量，$\mathbf{m}_1$ 和 $\mathbf{m}_2$ 表示力矩向量（表示为矢径的顺时针法线）。最后，假设碰撞法线定义为 $\mathbf{n}$。据此，可通过 4 个方程表达碰撞效果，如下所示：

$$m_1\mathbf{v}_1=m_1\mathbf{u}_1+J\mathbf{n}$$
$$m_2\mathbf{v}_2=m_2\mathbf{u}_2-J\mathbf{n}$$
$$I_1\varphi_1=I_1\omega_1+J\mathbf{m}_1\cdot\mathbf{n}$$
$$I_2\varphi_2=I_2\omega_2+J\mathbf{m}_2\cdot\mathbf{n}$$

当给定上述方程的表达方式后，可假设碰撞点处的行为相同（针对两个点），且与对象的角速度无关。当然，读者也可对此进行微调。其中，两个碰撞点以某一相对速度运动，经计算后可得到 $\mathbf{u}_2-\mathbf{u}_1+\mathbf{m}_2\omega_2-\mathbf{m}_1\omega_1$。从碰撞角度来看，对象的其他部分皆处于未知状态。碰撞过程仅关注对象以相对速度运动，以及二者所包含的质量。最终，碰撞使得对象以相同方式反弹，也就是说，对象之间处于完全弹性碰撞状态，相对速度正交于碰撞行为并遵循下列方程：

$$(\mathbf{v}_2-\mathbf{v}_1+\mathbf{m}_2\varphi_2-\mathbf{m}_1\varphi_1)\cdot\mathbf{n}=-(\mathbf{u}_2-\mathbf{u}_1+\mathbf{m}_2\omega_2-\mathbf{m}_1\omega_1)\cdot\mathbf{n}$$

上式体现了能量守恒状态，这一点与线性碰撞类似。由于碰撞仅影响到法线方向，因此，若初始法向速度等于最终速度，则能量处于守恒状态。

结合上述 5 个方程，则可得到如下算式：

$$\left(\mathbf{u}_2-\frac{J}{m_2}\mathbf{n}-\left(\mathbf{u}_1+\frac{J}{m_1}\mathbf{n}\right)+\mathbf{m}_2\left(\omega_2-\frac{J}{I_2}\mathbf{m}_2\cdot\mathbf{n}\right)-\mathbf{m}_1\left(\omega_1+\frac{J}{I_1}\mathbf{m}_1\cdot\mathbf{n}\right)\right)\cdot\mathbf{n}$$

$$= -(\mathbf{u}_2 - \mathbf{u}_1 + \mathbf{m}_2\omega_2 - \mathbf{m}_1\omega_1)\cdot\mathbf{n}$$

合并相关数据项并做适当简化后，可得到如下算式：

$$2(\mathbf{u}_2 - \mathbf{u}_1 + \mathbf{m}_2\omega_2 - \mathbf{m}_1\omega_1)\cdot\mathbf{n} - \mathrm{J}\left(\frac{1}{m_2}\mathbf{n} - \frac{1}{m_1}\mathbf{n} + \mathbf{m}_2\left(\frac{1}{I_2}\mathbf{m}_2\cdot\mathbf{n}\right) + \mathbf{m}_1\left(\frac{1}{I_1}\mathbf{m}_1\cdot\mathbf{n}\right)\right) = 0$$

$$2(\mathbf{u}_2 - \mathbf{u}_1 + \mathbf{m}_2\omega_2 - \mathbf{m}_1\omega_1)\cdot\mathbf{n} - \mathrm{J}\left(\frac{1}{m_2} + \frac{1}{m_1} + \frac{1}{I_2}(\mathbf{m}_2\cdot\mathbf{n})^2 + \frac{1}{I_1}(\mathbf{m}_1\cdot\mathbf{n})^2\right) = 0$$

上式经整合后可计算 J。最后，可将 J 值代入至上述 4 个动量方程整合，进而计算新速度。

需要注意的是，若质量或转动惯量趋于无穷大，则上述方程将产生不同的计算结果——无论是在线动或是角动情况下，对象均处于固定状态。例如，若两个转动惯量均趋于无穷大，则对应项将从方程中消去，问题则演变为线性碰撞。随后，角速度不会受到冲量的影响。

非弹性碰撞的计算过程也不复杂，对此，只需将方程中的“2”替换为$(1 + e)$即可。其中，e 表示为回弹系数，其值位于 0（粘性物）～1（微粒）之间。

综上所述，resolveAngularCollision()函数的实现过程如下所示：

```
function resolveAngularCollision(obj1, obj2, n, mom1, mom2)
  set u1 to obj1.getVelocity()
  set u2 to obj2.getVelocity()
  set om1 to obj1.getAngularVelocity()
  set om2 to obj2.getAngularVelocity()

  set J to 2*dotProduct(u2-u1+mom2*om2-mom1*om1,n)
  set denom to 0
  if not obj1.fixedLinear() then
    set m1 to obj1.getMass()
    set denom to denom + (1/m1)
  end if
    if not obj2.fixedLinear() then
    set m2 to obj2.getMass()
    set denom to denom + (1/m2)
  end if
    if not obj1.fixedAngular() then
    set moi1 to obj1.getMOI()
    set dp1 to dotProduct(mom1,n)
    set denom to denom + (dp1*dp1/moi1)
  end if
    if not obj2.fixedAngular() then
    set moi2 to obj2.getMOI()
    set dp2 to dotProduct(mom2,n)
    set denom to denom + (dp2*dp2/moi2)
  end if
    if denom=0 then exit //coincident axes or other weirdness
    set J to J/denom
    if not obj1.fixedLinear() then obj1.setVelocity(u1+J*n/m1)
    if not obj2.fixedLinear() then obj2.setVelocity(u2-J*n/m2)
    if not obj1.fixedAngular() then obj1.setAngularVelocity(om1+J*dp1/moi1)
```

```
    if not obj2.fixedAngular() then obj2.setAngularVelocity(om2-J*dp2/moi2)
end function
```

如前所述，resolveAngularCollision()函数假设两个对象的速度可直接予以调整。

13.5　向撞球游戏中加入旋转行为

出于完整性考量，可将旋转行为加入至第 11 章所讨论的撞球游戏中。除了定义运动向量之外，还需进一步确定从球心左侧或右侧击打球体。在击打球体的过程中，作用力偏移质心位置，因而将生成角动量。此处，也可通过基于中心位置的线性距离函数考察角动量。

【提示】旋转有时也称作侧旋，在当前上下文环境中，旋转行为意指侧旋。术语“侧旋”源自法语单词“角度（angle）”，特指几何角度。当发声重音变化时，该单词也具有“侧旋”的含义。

在撞球游戏中，由于球体一般采用圆形表示，因而角运动不会对碰撞检测产生影响，但会影响到碰撞处理方案，其原因在于，前述假设条件并未在撞球游戏中体现。考虑到库边、桌面与球体之间的摩擦力，碰撞对象的冲量沿碰撞法线予以作用。这也意味着，球体的部分角动量将转化为线动量，反之亦然。

图 13.13 显示了球体和库边之间的碰撞过程。这里，假设碰撞过程包含两部分内容。首先，角速度减少 ϕ，对此，需在碰撞点处沿库边存在一个冲量 J，并通过 $I\phi = Jr$ 计算 J。随后，可使用该冲量计算最终的线速度变化，如下所示：

$$m\mathbf{v} = m\mathbf{u} + J\mathbf{t}$$

$$\mathbf{v} = \mathbf{u} + \frac{I\phi}{m}\mathbf{t}$$

待最新速度计算完毕后，则可通过常规方式应用垂直冲量。

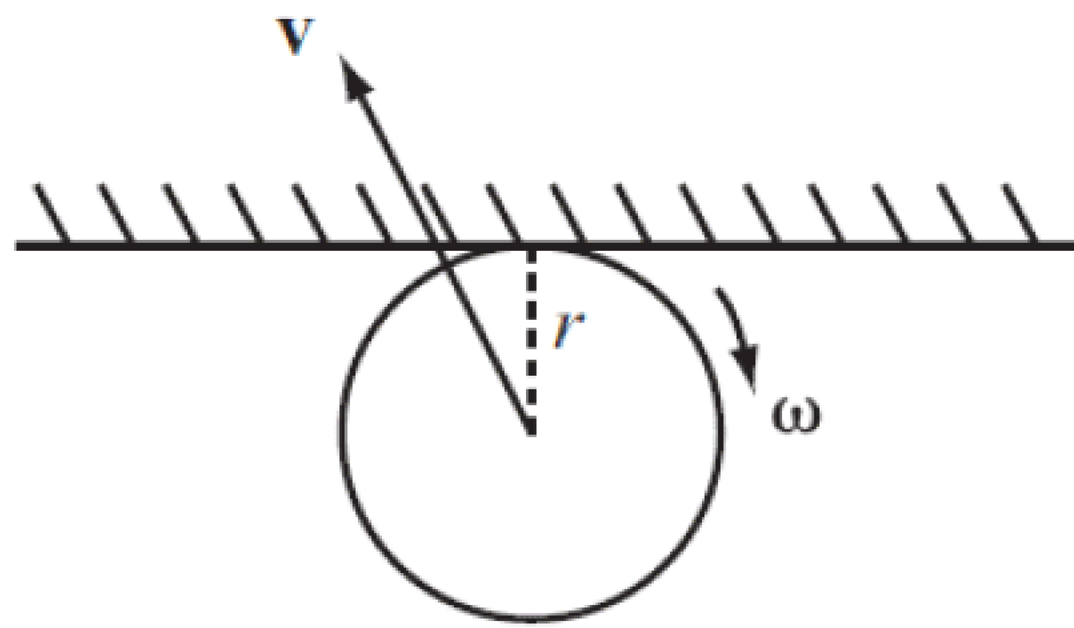


图 13.13　粘滞表面的碰撞行为

13.6　本 章 练 习

【练习 13.1】试编写 resolveCushionCollision(obj1, obj2, normal, moment, slow)函数，该函数

计算与库边之间的碰撞结果，并移除了某些角速度常量值。

另外，鉴于缺少角速度比率，该函数还需进一步计算角速度和线速度。

【练习 13.2】试编写函数以计算运动、旋转正方形与库边之间的碰撞行为。

13.7　本 章 小 结

从本质上讲，角运动相对简单，但会涉及较多的细节内容。另外，其实现过程也较为复杂。通常，游戏设计人员应对此加以谨慎处理。本章讨论了碰撞系统的实现方案，并引入了旋转行为。至少，读者当前可解决圆和直线之间的碰撞问题。第 14 章将对此进行扩展，并考察旋转与摩擦力关联方式。

至此，读者应掌握如下内容：

- 杠杆的运动和平衡方式。
- 转矩的含义及其与作用力的关系。
- 角动量和能量与线性对应项之间的相似度。
- 如何计算某些简单旋转图形的碰撞点。
- 如何处理一个或多个旋转对象之间的碰撞行为。

第 14 章　摩　擦　力

本章包含如下内容：

- 概述。
- 摩擦力的工作方式。
- 摩擦力和角运动。

14.1　概　　述

第 13 章初步探讨了旋转问题，本章将继续讨论这一重要的主题。前述内容主要关注撞球游戏中的侧旋现象，且未涉及较为常见的上旋和下旋行为。相应地，上旋和下旋通常称作滚动旋转。这里的问题是，如何定义球体的滚动旋转？毕竟，当球体常规运行时，通常不存在滚动旋转。实际上，第 13 章所讨论的动量计算则完全省略了侧旋行为。

当球体滚动时，其表面与其他物体表面相接触并产生摩擦力。当两个对象碰撞并反向运动时，该过程也会产生摩擦力。前述章节曾对这一概念有所提及，并解释了侧旋球体与库边碰撞后为何以某一角度回弹。本章将对该现象进行深入讨论。

14.2　摩擦力的工作方式

摩擦力主要源自对象的不规则表面。针对砖面和大理石表面，前者较为粗糙，而后者相对光滑，因而砖块具有较大的摩擦力。若在显微镜下进行观察，则会发现表面上含有大量的凹陷和突起。当此类表面彼此碰撞时，将会产生作用力并沿切向作用于碰撞平面上。本书第 26 章还将讨论碰撞过程中的瞬时摩擦力，本章则探讨一类传统的摩擦力，此类摩擦力持续作用于运动对象上。

14.2.1　摩擦系数

由于摩擦力随正交方向上的作用力而变化，因而摩擦力并非是一种常规的作用力。实际上，摩擦力正比于法向作用力。针对任意两个物体，存在一个摩擦系数 μ 并满足如下关系：

$$摩擦力=\mu\times正交作用力$$

图 14.1 显示了摩擦力示意图，其中，重量为 W 的盒体沿角度为 θ 的斜面运动。图中，盒体的重力向下作用，因而垂直于斜面的作用力为 $W\cos\theta$。据此，摩擦力与运动方向相反，且大小为

$\mu W\cos\theta$。最终，平面切向的合力表示为 $W\sin\theta - \mu\cos\theta = W(\sin\theta - \mu\cos\theta)$。

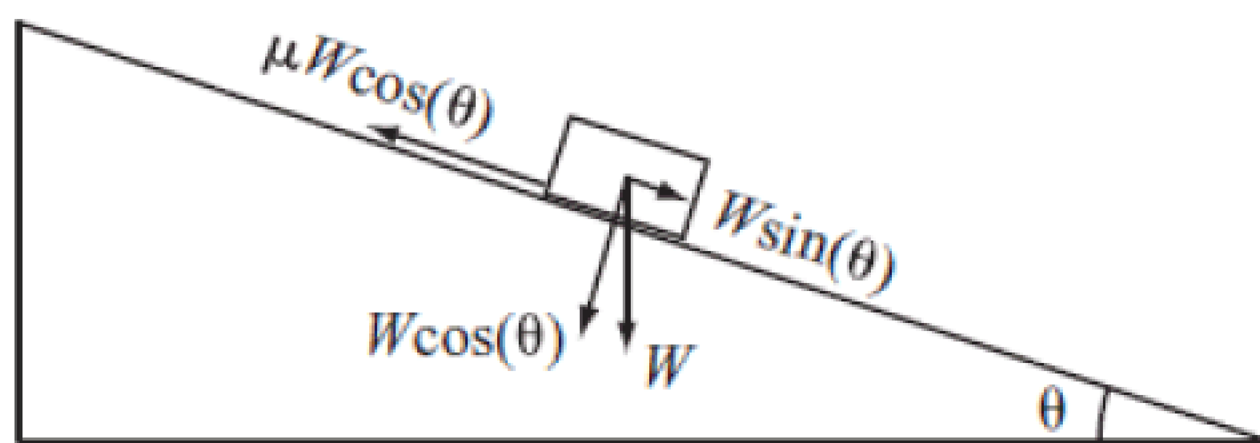


图 14.1　斜面上的摩擦力

在图 14.1 中，根据牛顿第三定律，盒体也将承受一个方向相反、大小为 $W\cos\theta$ 的作用力，以使其在平面法线方向上处于平衡状态。摩擦力即源自反作用力。另外，若平面包含某一角度，在基于重力的作用力抵销摩擦力后，盒体将处于运动状态并受到摩擦力作用。最终，盒体以恒定速度运动，这也是摩擦系数的测量方法。针对抵消摩擦力所需的角度值，可根据下列方式予以计算：

$$W(\sin\theta - \mu\cos\theta) = 0$$

$$\sin\theta = \mu\cos\theta$$

$$\tan\theta = \mu$$

若 θ 小于 μ，则盒体运动逐渐缓慢并趋于停止；若 θ 大于 μ，则重力克服摩擦力且盒体处于加速状态。

截止到目前为止，前述讨论可视为一类简化版本，且仅描述了动摩擦系数。除此之外，还存在一种静摩擦系数。当对象处于静止状态并承受下滑力时，对象的静摩擦力将抑制对象的运动。

动摩擦系数常记为 μK，而静摩擦系数则记为 μS。由于静摩擦力体现了一类最大可能作用力，而非实际值，因而与动摩擦力相比，其计算过程稍具难度。若作用力未突破静摩擦力极限，则对象依然会处于静止状态，且等值大小的作用力将抵消所谓的运动力。

通常情况下，静摩擦系数（μS）大于动摩擦系数。也就是说，与加速已处于运动状态的物体相比，打破其静止状态往往需要更大的作用力。在图 14.1 中，假设盒体静止于斜面上，则静态摩擦力可表示为 $\mu SW\cos\theta$，即作用于静态物体上的、最大可能摩擦力。若基于盒体重量的作用力小于该值，则盒体保持静止。换而言之，若 $\tan\theta < \mu S$，则盒体对象处于静止状态。其中，θ 值称作临界角。

最后，斜面也包含一个角度范围，并满足下列情形：若盒体静止，则斜面也处于静止状态。同样，该结论对于运动情形同样成立。据此，有 $\mu K < \tan\theta < \mu S$。

综上所述，resultantForceOnObject()函数的实现过程如下所示：

```
function resultantForceOnObject(nonFrictionalForce, velocity,coefficient)
  set tang to norm(velocity)
  set norm to normalVector(tang)
  set normalForce to component (nonFrictionalForce, norm)
  set tangentialForce to component (nonFrictionalForce, tang)
  set frictionalForce to coefficient * magnitude(normalForce)
  if frictionalForce>tangentialForce and magnitude(velocity)>0
  then return vector(0,0)
  otherwise return (tangentialForce - frictionalForce) * tang
end function
```

针对静摩擦力和动摩擦力，resultantForceOnObject()函数工作良好。对于不同类型的摩擦力（取决于当前对象处于运动状态抑或是静止状态），可采用正确值替换摩擦系数。另外，参数 nonFrictionalForce 表示为作用于当前对象上的其他作用力，而非对象与平面之间的交互作用力。函数的返回值表示作用于对象上的合力（包括摩擦力和基于牛顿第三定律的反作用力），且平行于当前平面。当然，读者还可尝试编写该函数的特定版本，进而处理对象沿斜面下滑这一情形。

14.2.2 摩擦力和能量

若将摩擦力引入至力学仿真环境中，则相关问题涉及能量和动量守恒。对于静态对象，动量守恒并非是重点考察内容，而能量损失主要体现于两个对象碰撞过程中的热能。

通过考察做功情况，可计算基于摩擦力的能量损失。作为一个能量术语，“功”表示执行任务过程中所消耗的能量，且通常使物体处于运动状态，其计算方式如下所示：

$$\text{功}=\text{作用力}\times\text{移动距离}$$

上式表明，作用力越大，则针对特定的距离，所消耗的能量也就越大。需要注意的是，若物体未产生移动，无论作用力大小如何，该过程的做功均为 0。功可视为能量的一种有效表达方式。

“功”的日常描述并不关心对象是否处于运动状态。例如，若读者尝试推动一面静止的砖堆，能量消耗仅来自读者自身，且砖墙的位置未产生任何变化。从物理学角度来看，该过程的做功为 0。能量消耗主要体现于肌肉间的化学反应。该过程涉及静摩擦力，此类摩擦力对于能量计算不会产生任何影响。

然而，若对象处于运动状态且受到动摩擦力的作用，则可计算摩擦力功耗，且通常为移动距离的固定倍数，如下所示：

$$\text{功耗}=\text{摩擦力}\times\text{距离}=\mu\times\text{法向作用力}\times\text{距离}$$

上式的复杂之处在于，运动距离同样为作用力的函数。此处，假设对象沿桌面运动，例如冰球。在特定的时刻 t，冰球的运动距离为 $ut+\frac{1}{2}at^2$，根据牛顿第二定律，能量损失为 $Fut+\frac{F^2t^2}{2m}$。由于摩擦力与冰球的重量 mg 相关，因而上式最终可表示为：

$$\text{能量损失}=\mu mg\left(ut+\frac{\mu gt^2}{2}\right)$$

另外，速度损失呈线性递减状态，因而易于计算，如下所示：

$$\text{速度损失}=\mu gt$$

这一结论可在前述撞球游戏中得到证实。然而，球体运动远复杂于盒体运动，因而有必要重新审视当前模拟环境。

14.2.3 空气阻力和临界下降速度

空气阻力（更为通用的称谓则是流体阻力）则是另一种较为常见的摩擦力。鉴于液体和气体具有相似的特征，因而此处“流体”泛指液体或气体。前述内容曾定义了摩擦力，进而体现了一

种不同方式的作用力。物体在空气中下落时，将受到来自空气分子的碰撞，进而生成阻力，该碰撞类型与前述碰撞讨论不同（即盒体在地面运动时的滑动力）。

与摩擦力相比，阻力计算则稍显复杂，其中一个较为重要的数据值则是阻力系数，常记为 CD。相应地，阻力方程往往表示为如下形式：

$$阻力=\frac{1}{2}CD\,\rho\,v^2A$$

其中，希腊字母 ρ 表示为流体压强，A 表示为流体的前向面积，v 则表示为流体的运动速度。关于压强和面积值，鉴于当前问题较为简单，因而此类数据项均为常数。此处需要关注的问题是，阻力正比于速度的平方。当然，读者也可自定义阻力系数，并记为 μD，如下所示：

$$阻力=\mu_D v^2$$

【提示】另一个需要注意的数据项是压强。这里，压强体现了流体分子的密集程度，该值越高，则落体将会受到更多的碰撞。类似地，物体的表面积越大，下落时将会撞击更多的分子，这也是降落伞的下落速度远慢于一颗豌豆的主要原因。考虑到与接触面积相关，因而可按照相同方法划分摩擦系数。

由于阻力在某一阶段与速度相关，因而落体将到达某一极限速度，此时，阻力等于该物体所受重力。若以该速度运行，则物体将停止加速行为。该速度称作临界速度，其计算方式如下所示：

$$mg=\mu_D v^2$$

$$v=\sqrt{\frac{mg}{\mu_D}}$$

由于 μ_D 正比于面积值，因而临界速度反比于面积的平方根。进一步讲，圆形或球体表面积正比于半径的平方值，因此，临界速度与半径之间呈反比关系。例如，若降落伞的半径加倍，则临界速度随之减半。

14.3　摩擦力和角运动

阻力的实现过程并不复杂，前述撞球游戏也对此有所讨论。除此之外，还需对其他复杂问题予以考察。

14.3.1　轮胎和牵引力

机动车辆大多配有轮胎，作用力可使轮胎处于运动状态。当车辆运动时，柱状对象中的线性作用力可转换为转矩，并作用于轮轴上。当轮胎位于地面上时，为何不会出现侧旋打滑现象（类似于深陷泥地中的车辆）？答案在于摩擦力，摩擦力可使车辆移动。当摩擦力通过此方式作用时，该作用力常称作牵引力或抓地力。

抓地力一般为静摩擦力。当轮胎与地面接触并试图转动时，通常会受到来自地面的静摩擦力的作用，并抵消轮胎上的转矩，进而使得轮胎滚动而非侧旋。因此，摩擦力使得车辆前向行驶，

并与运动方向保持一致。实际上，车辆的运动行为可视为牛顿第三定律的结果，也就是说前向作用力可视为地面轮胎的反作用力。由于道路不会回退，因而车辆一路前行。

这里，可计算与地面接触点处的作用力。若轮胎所受转矩为 T 且半径为 r，则轮胎表面点处的作用力可表示为 $\frac{T}{2r}$。若该作用力大于轮胎与地面间的静摩擦力，情况又当如何？对此，轮胎将在地面上呈滑动状态，但车辆仍保持原地不动。若情况与此相反，则车辆急停并留下明显的刹车痕迹。因此，安全驾驶的主要任务即是对轮胎的转矩实施有效的操控。

齿轮传动装置可有效地控制转矩级别。当踩下油门后，燃油的供给量增加，功率或每秒消耗的能量也将随之增加。这里，功率可通过不同方式进行转化。类似于功可表示为作用力与距离的乘积，功率可通过转矩和角速度加以定义，如下所示：

功率=作用力×速度=转矩×角速度

当增加功率时，作用力/转矩也随之增加——与此对应的是，加速度或速度也将增加。齿轮传动装置可实现相应的调节功能，低速档可获得较大的转矩以及加速度，但最大速度较低；而高速档则具有较小的加速度以及较大的速度。因此，通常采用低档提速，并于随后切换至高速档以实现高效的巡航行驶，这也是为何使用低速档超车的原因。相应地，当摆脱轮胎打滑这一类困境时，则可换至高速档位，降低转矩并增加地面的抓地力。

刹车过程与加速行为十分类似，但效果则截然相反。当车里处于制动状态时，将获得轮胎上的反向转矩。当然，驾驶员并不希望全部锁死轮胎，这将生成较大的静摩擦力，并导致车辆打滑。相反，轮胎应处于可控状态下，进而保持车辆与路面之间的抓地力。

对于撞球游戏，当球体（或与球杆）之间产生正面撞击时，球体于开始阶段无侧旋运动，类似于前述牛顿摆。然而，该过程并不会持续太久。在桌面滑行过程中，球体受到动摩擦力的影响，这将降低球体的整体运动速度，当侧旋球体的表面速度等于球体的当前速度时，球体将不再滑动。相反，球体像轮胎一样处于滚动状态。

此时，静摩擦力将起主导作用。与动摩擦力不同，静摩擦力并不会降低球体的速度，且球体的速率将发生变化。这一变化通过空气阻力、粘性碰撞以及影响滚动速度的动摩擦力所导致的能量损失加以描述。关于动能，在球体运动的每一时刻，物体挤压布料的纤维，这将导致产生源自内部的阻力。

最终结果可描述为，球体在桌面上沿运动方向前向侧旋，此类侧旋称作上旋。在相反方向上，则称作后旋。通过侧旋行为，当选手采用偏心（冲量）方式击打母球时，即可打出上旋球或后旋球。

除此之外，当前模拟环境还包含了其他复杂项，其中，上旋或下旋使得球体运动类似于轮胎。如果球体的上旋与其自然滚动相比过于强烈，则摩擦力将指向前方而非后方，并在侧旋的同时加速球体运动。相反，若侧旋足够强烈，基于后旋的摩擦力可降低球体运行速度，并使其以相反方向运动。

当处理碰撞行为时，上述行为均清晰可见。当与另一球体发生碰撞时（该球体具有较低的摩擦系数），上旋通常不会受到影响。最终结果可描述为，即使处于回弹阶段，球体依然向前滚动。此类侧旋类似于轮胎，使得球体保持同一方向运动。若球体在碰撞时刻后旋，则将增加其回弹力度。

当读者体验撞球游戏时，可尝试上述碰撞行为。当某一球体沿速度向量直线击打另一球体时，第一个球体即刻停止，而第二个球体则以相同速度运动，这一点与牛顿摆十分类似。另外，若球杆击打球体上方，则该球体与其他对象碰撞后将短暂停止，并于随后前向滚动；相反，若在底部击打球体，则该球体停止并后向滚动。

14.3.2　摩擦力和打滑现象

不仅轮胎依靠摩擦力运动，对于人类的行为驱动，摩擦力同样不可或缺。当人类在地面上行进时，需要通过脚步施加后向作用力，而地面通过前向摩擦力与其实现反馈。

若人类可迈步前行，则脚步与地面之间须存在抓地力（牵引力），前提条件是，摩擦系数不应过小。类似于轮胎侧旋现象，若脚步施加的后向作用力克服了静摩擦力，则可视为作用于身体上的转矩。最终，这将导致侧旋现象，因而出现滑动或跌倒现象。对此，存在多种技巧可防止这一危险行为，例如挥舞手臂以提供“抵消”转矩。该动作类似于直升飞机，并通过副螺旋桨克服主螺旋桨导致的旋转问题。若全部辅助方法均无效，则可尝试不同的运动策略。若不采用摩擦力，可通过平衡方法予以调整。例如，可向前移动腿部并与另一条腿平衡。四蹄动物常利用该方法实现自身的平衡。

通常情况下，净转矩往往会导致对象处于跌落状态，如图 14.2 所示。其中，两个对象受到源自地面的向上作用力，以及基于自身重量的向下作用力，且重力穿越对象的重心位置。在第一个示例中，重力穿越对象与地面之间的碰撞点 P，也就是说，净转矩为 0。在第二个示例中，重力则位于碰撞点外部，这也意味着，点 Q 表示为支点。

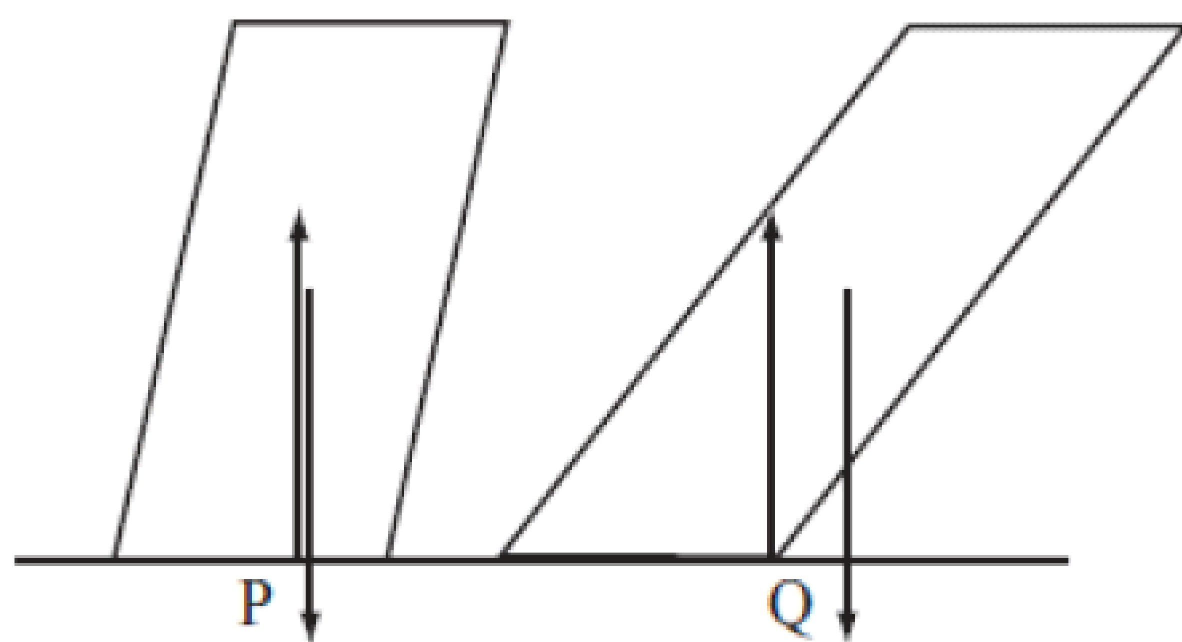


图 14.2　跌落过程

在行进过程中，人类可通过自身的平衡感控制作用力，例如扭动脚部以改变与地面之间的碰撞点，或者调整手臂以及臀部位置。当处于跌落状态时，还可通过另一只脚抵住地面以获得相应的支撑力。

【**提示**】针对图 14.2 中的第一个示例，由于中心并未位于碰撞直线中心上方，因而一侧存在净作用力，进而导致出现转矩。然而，鉴于重力穿越中心位置，因而反作用力被抵消，并可视为点 P 处的、方向向上的点作用力。对应效果类似于使用两个支架台支撑一个平台。其中，重量分布于两个支点处。在当前示例中，支架台被颠倒搁置。

图 14.3 显示了如何施加作用力以增加摩擦力效果。在跑动过程中，运动员将受到来自地面的较大转矩以防止侧旋。同时，运动员可前向倾侧身体并产生逆转矩。由于腿部大部分力量均朝

向地面直线，因而可向其施加较大的作用力。

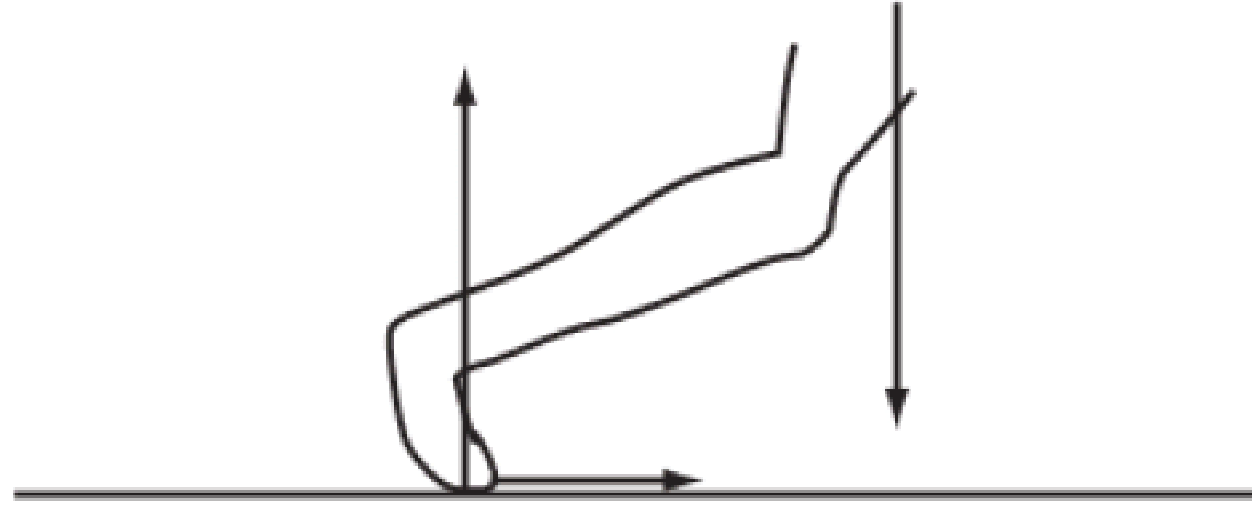


图 14.3　使用运行时的摩擦

14.4　本 章 练 习

【练习 14.1】试编写 applyFriction()函数，对应参数为 velocity，topSpin，Radius，mass，muK，muS 以及 time。另外，该函数还应考虑摩擦力效果，进而计算撞球的运动行为。

该函数运用球体的当前状态，并返回最新的速度和旋转状态。针对摩擦系数，此处假设摩擦力在短时间内保持恒定。

14.5　本 章 小 结

本章讨论了力学中某些并不常见的话题，读者应深入理解其工作方式。据此，本章还提供了少量方程以及通用代码。

本章重点分析了摩擦力的工作方式以及不同场合下的应用方式，并考察了撞球游戏这一开发环境。第 15 章将讨论另一个补充话题，并对能量和动量进行归纳性总结。除此之外，第 15 章还将对动量和张力加以分析。

至此，读者应掌握如下内容：

- 摩擦力及其计算方法。
- 摩擦系数的含义以及静摩擦力和动摩擦力之间的差别。
- 阻力及其计算方法。
- 摩擦力与线动量和角动量之间的转换方式。
- 运动对象的跌落现象。

第 15 章　绳索、滑轮和传送带

本章包含如下内容：

- 概述。
- 拉动对象。
- 不可扩展的绳索。
- 连续动量。

15.1　概　　述

本章将考察与能量和动量计算相关的某些特殊实例，且均涉及连接对象。在 15.2 节中，两个对象将通过一条“理想化”的绳索连接在一起，并通过滑轮方便地提升对象。后续内容还将讨论基于连续质量变化的动量变化示例，例如传动带上的重物以及火箭所携带的燃料。

15.2　拉 动 对 象

若绳索绑定于对象一端，则可通过拉动该绳索移动对象。施加于绳索上的作用力沿绳索对另一端的对象产生影响，该作用力也称作张力。

15.2.1　不可扩展的绳索

此处将讨论一类理想化的对象，即轻质不可扩展的绳索，并与前述所讨论的杠杆类似。此类对象不包含质量且无弹性可言（第 16 章将对弹性予以介绍）。除此之外，本章所讨论的绳索也不会出现彼此缠绕这一现象。

轻质不可扩展绳索的长度定义为 l，当采用此类绳索连接两个对象时，对象间的距离不大于 l，且二者可彼此靠近。若绳索处于拉紧状态，则将受到张力的作用。如图 15.1 所示，张力作用于两个方向，并位于绳索上的各点处。特别地，绳索一端的对象将受到源自绳索的相同作用力 T。

若拉动绳索一端，则向其施加作用力并生成张力。需要说明的是，张力对牛顿第三定律进行了适当的扩展，若向绳索施加某一作用力，则绳索也向施加者赋予相同的作用力，最终结果可描述为，作用力“传输”至另一对象。当然，若拖曳绳索并突然停止（而非持续拉动该绳索），则可使对象处于运动状态，而该绳索则最终处于松弛状态。

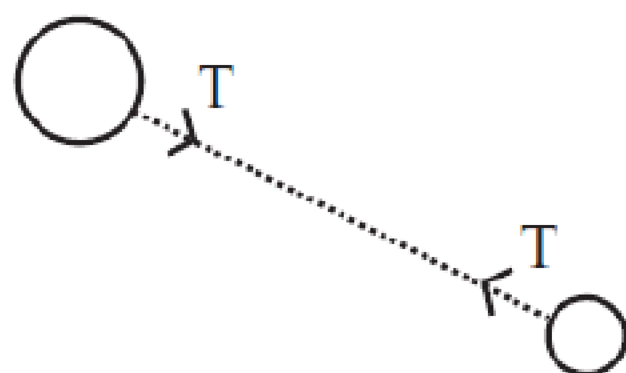


图 15.1　张力作用下的两个对象。这里，张力源自处于拉紧状态的绳索

针对理想化的绳索，其长度不会发生任何变化，但此类绳索依然存在某些特点值得研究。第 16 章将对此予以介绍，并讨论绳索与钟摆之间的关系。

15.2.2　桌面上的绳索

根据物理学视角，基础型绳索并无太多特色，但考虑到可改变作用力方向，此类绳索依然存在应用价值。如图 15.2 所示，某一绳索连接两个盒体对象，其中一个盒体对象悬挂于桌面外侧。尽管处于弯曲状态，但绳索依然提供单一作用力。

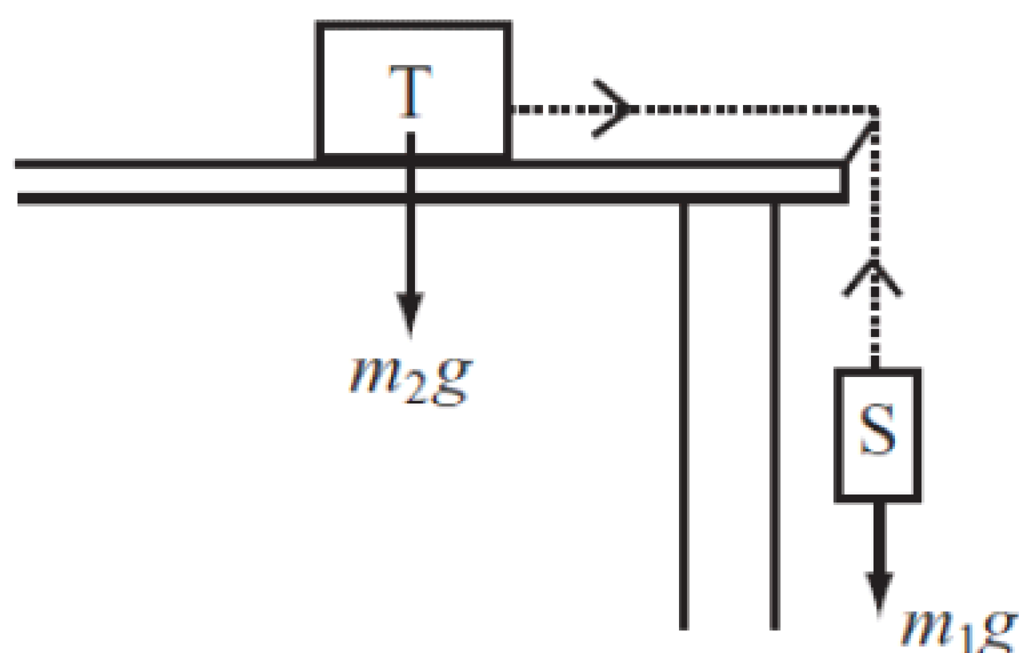


图 15.2　悬挂于桌面外侧的绳索

在图 15.2 中，悬挂盒体 S 受到基于自身重量的作用力，这将在绳索中产生张力。相应地，张力通过绳索传输，并向盒体 T 施加相同的作用力。在张力作用下，盒体 T 处于加速状态。除此之外，由于绳索不可扩展，因而两个盒体应以相同速率加速。根据牛顿第二定律，这将生成如下两个方程：

$$\text{张力} = m_2 a$$

$$m_1 g - \text{张力} = m_1 a$$

若消去上式中的“张力”一项，则有$(m_1 + m_2)\,a = m_1 g$。换而言之，两个对象均承受 S 重力，并同时处于加速状态。

此处，还可对摩擦力进行查看。盒体 T 受到源自桌面的摩擦力，且仅取决于重量 T。若盒体对象处于静止状态，即盒体 S 处于平衡状态，因而绳索张力等于盒体重量。据此可知 $\mu_s m_2 g \geqslant m_1 g$，因而有 $\mu_s m_2 \geqslant m_1$。同样，若不等式为 false，则盒体对象无法保持静止平衡状态。

15.2.3　绳索和圆周运动

图 15.3 显示了绳索的另一种情形，其中，对象绑定于绳索一端，并围绕中心位置转动，即

圆周运动。

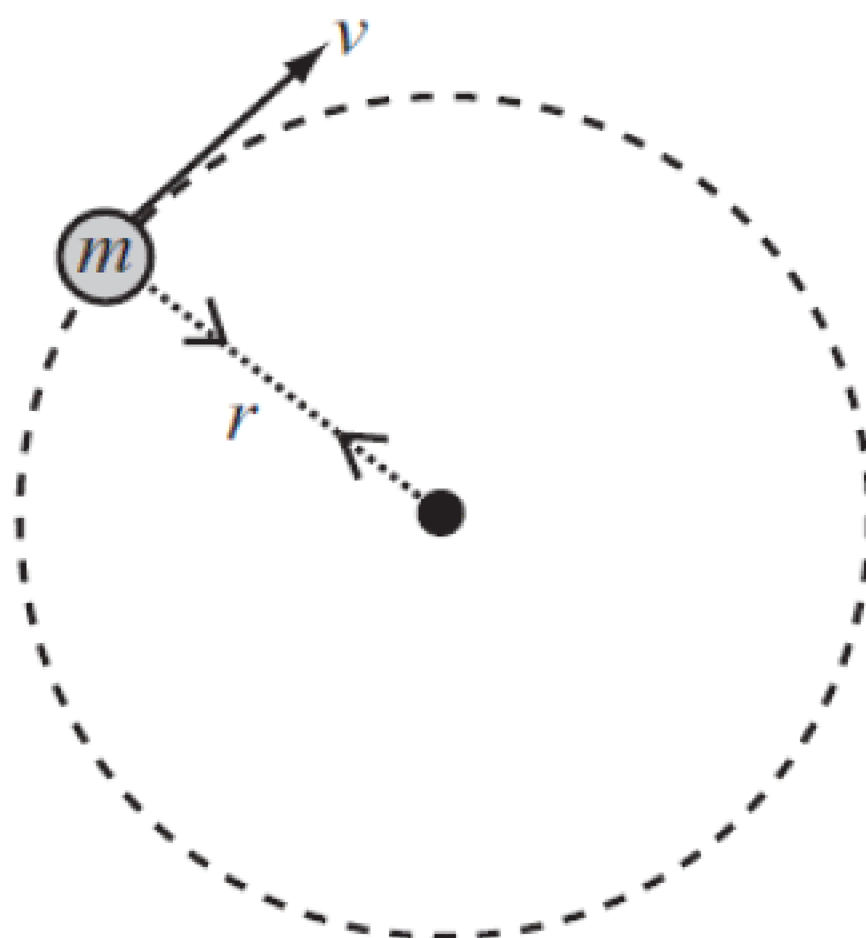


图 15.3　基于绳索的圆周运动

在第 12 章中曾有所提及，若对象做圆周运动，则需要向圆心位置提供向心力。这里，绳索通过张力提供向心力，并将其施加于绳索持有者。随后，绳索持有者则会感受到明显的离心力。该作用力可通过$\frac{mv^2}{r}$予以计算，其中，v 表示为速度，r 表示为圆半径（此处为绳索长度），m 表示为轨道运动对象的质量。

在图 15.4 中，绳索穿过桌面，且桌面上的球体 T 以速度 v 旋转。若于随后释放空洞下方的球体 S，将施加向下作用力 m_1g。与此同时，球体 T 所施加的作用力为$\frac{m_2v^2}{r}$，其中，r 表示为空洞与球体 T 之间的绳索长度。

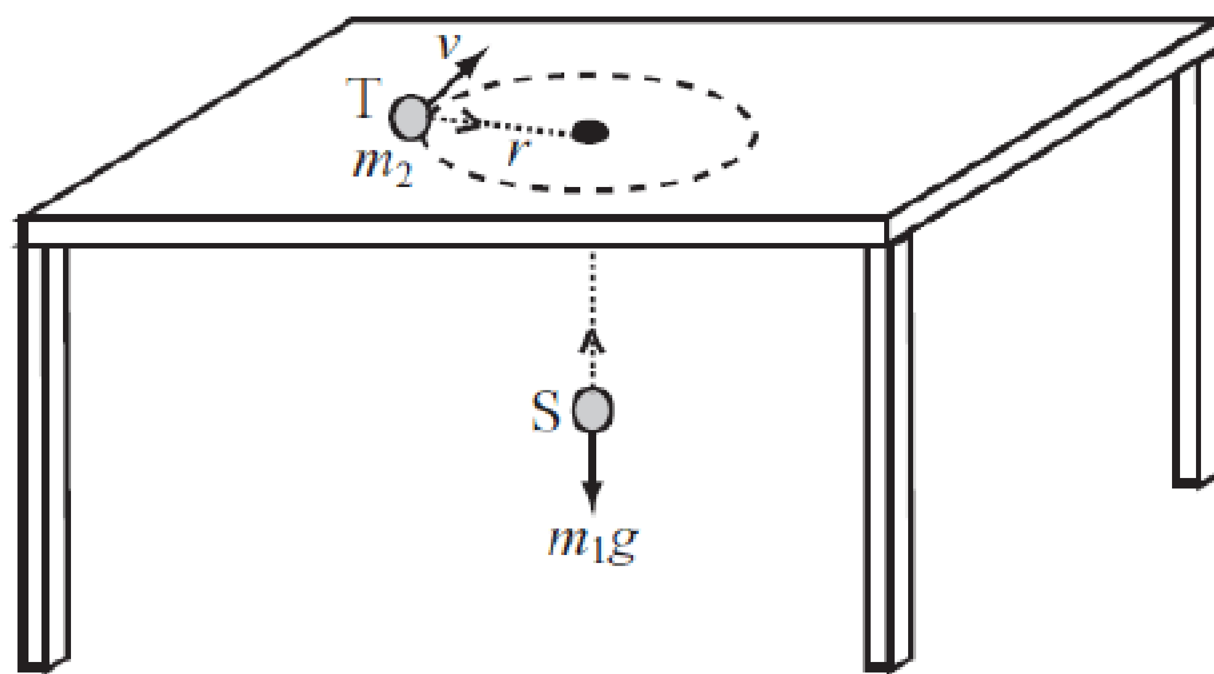


图 15.4　绳索穿过桌面

若源自 S 的作用力大于源自 T 的作用力，则球体通过空洞下沉。下沉过程将减少 r，鉴于角动量须保持守恒，因而 v 应适当增加。最终结果表明，半径为$\frac{k}{r}$且速度为 kv（$k>1$），进而生成新的作用力$\frac{km_2v^2}{r}$。在球体的持续下降过程中，基于圆周运动的作用力将不断增加，而源自重量的作用力则保持不变。最终，两个作用力在某一稳定轨道处处于均衡状态，且有$\frac{m_2v^2}{r}=m_2v$。

此类轨道的数量趋于无穷多个，且与 T 的初始角动量有关。在实际操作过程中，摩擦力可快速降低旋转球体的运动速度，读者可清晰地观察到这一现象。图 15.4 中的示例展示了一类较

好的处理方案，并可应用于引力轨道中。实际上，从数学角度上讲，针对穿越空洞的球体 S，其行为类似于重力在广义相对论中的表现，其中，对象的重力场根据其质量呈现为曲面空间。

15.2.4 滑轮

在真实世界中，绳索的主要应用之一即是滑轮组，图 15.5 显示了一类简单的滑轮装置，此类滑轮可视为前述“桌面盒体”方案的变本，并可改变作用力的方向。当采用滑轮装置时，提升物体所需要的效能与不使用绳索时一样，唯一的差别在于，搬运人员不必攀爬至高处完成此项任务。

图 15.6 显示了一类更为有效的滑轮装置，并包含了两个滑轮。其中，滑轮一负责提升物体，滑轮二负责固定支点位置。除此之外，绳索的一端绑定于对象上方的支点。据此，滑轮的操作方式产生了显著的变化。当前，绳索的两端均施加作用力。具体而言，绳索相对松弛一端施加作用力 F，而天花板处的支点则施加对应的反作用力 F。最终，对象上的合力为 $2F$，即上升力加倍。

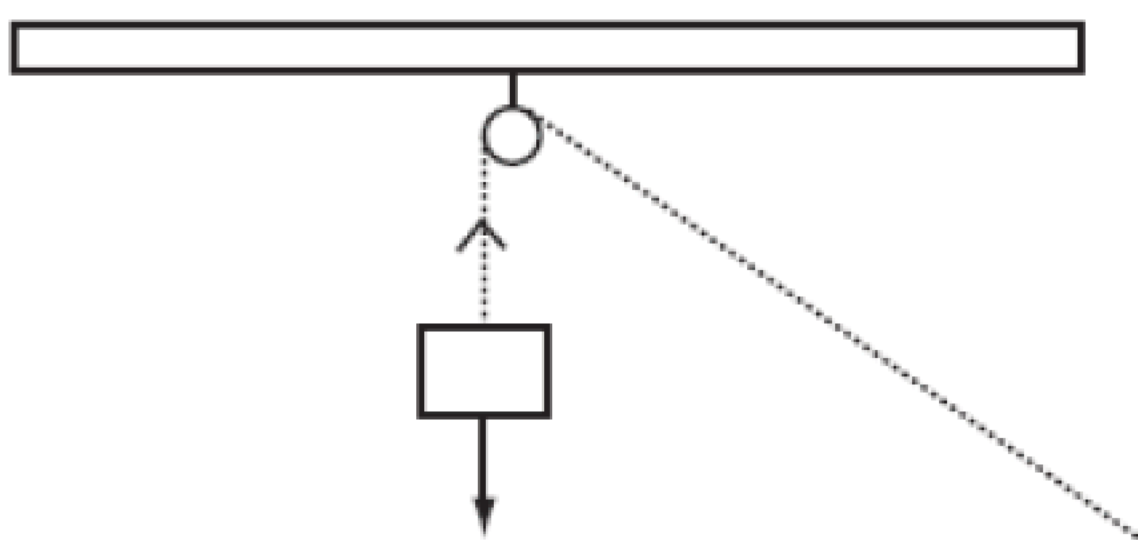


图 15.5　简单的滑轮装置

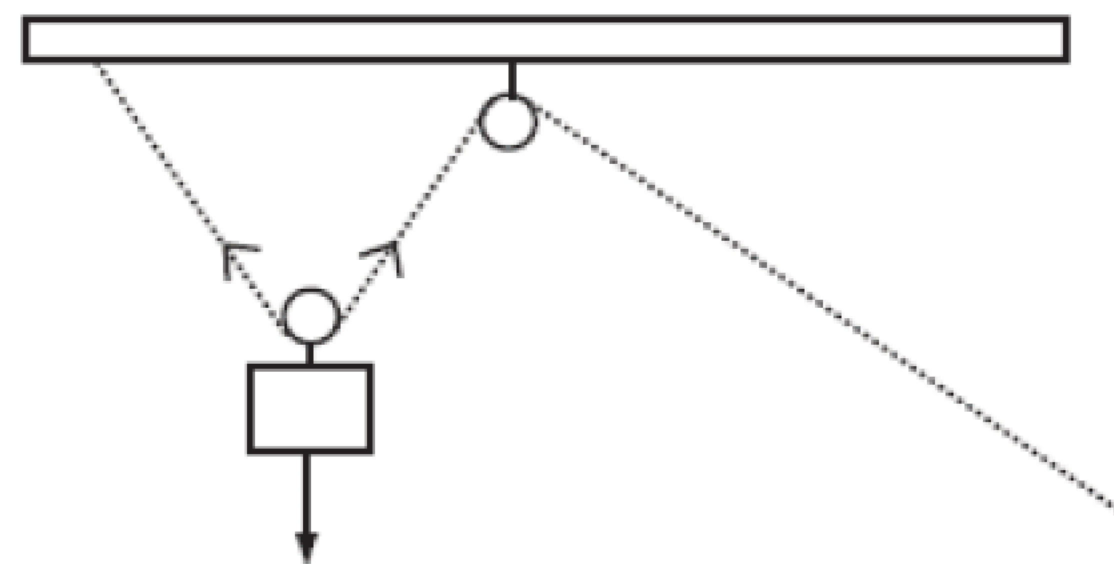


图 15.6　双滑轮机制

上升力加倍体现了某种矛盾，无论滑轮的数量如何，将物体提升既定距离所需的能量保持不变。对此，可考察操作过程中所消耗的功。此处，虽然需要两倍的作用力提升物体，但物体的移动距离减半。相应地，提升相同距离的物体则需要两倍的绳长，但该过程的功耗保持不变。

如不考虑滑轮间的摩擦力，待添加更多的滑轮后，所施加的作用力也随之降低。图 15.7 显示了 4 滑轮的工作状态，其中，绳长有所增加，但作用力随之降低，且功耗保持不变。滑轮组类似于齿轮传动机制，后者通过作用力换取距离或速度方面的收益。

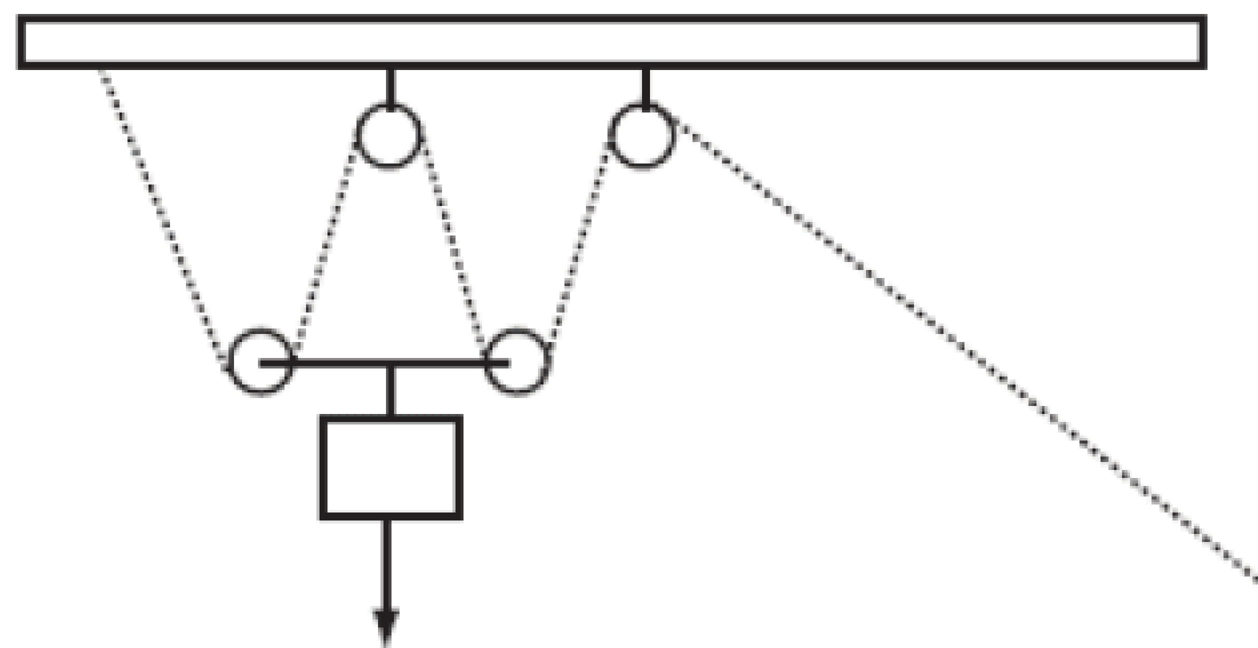


图 15.7　4 滑轮机制

【提示】在实际应用过程中，图 15.7 所示滑轮组常会处于不稳定状态，因而底部滑轮组常被替换为独立滑轮。

15.3　连续动量

第 7 章曾讨论了质量处于变化状态（燃料消耗）下的火箭的运动行为，具体内容尚未深入讨论。这里将再次对该问题进行回顾，即质量减少时对象的计算方式。

15.3.1　传送带

下面首先讨论相反情形，假设以每秒 k 千克的速率向传送带上添加细小颗粒物质，例如沙粒，其中，沙粒以较为缓和的方式加载且不会产生滚动。若传送带以固定速度 v 前进，试计算传动装置的功率。

除了暂且忽略传送带上的摩擦力之外，传送带的长度也不会产生任何变化。当沙粒处于运动状态时，鉴于牛顿第一定律，保持该状态并非难事。这里的问题是，加载沙粒时所需的能量消耗。在每秒中，k 千克沙粒加速至速度 v，因而其动能从 0 转化为$\frac{1}{2}kv^2$。这也意味着，传送带的功率须为$\frac{1}{2}kv^2$。

待功率计算完毕后，作用力计算则相对简单。鉴于功率=作用力×速度，因而作用力可表示为$\frac{1}{2}kv$。

相同计算也适用于自动扶梯。自动扶梯与传送带十分类似，后者在水平方向上传送物体，而自动扶梯则将物体传送至某一高度处。由于此处需考虑自动扶梯的长度，因而该计算过程稍显复杂。

图 15.8 显示了速度为 v、宽度为 l 且高度为 h 的自动扶梯。在真实世界中，扶梯上的对象并非均匀分布，且人们往往会跨步站立于扶梯上。在当前计算中，假设扶梯满载，且图中人物角色以每秒 k 千克的恒定速率沿扶梯行进。

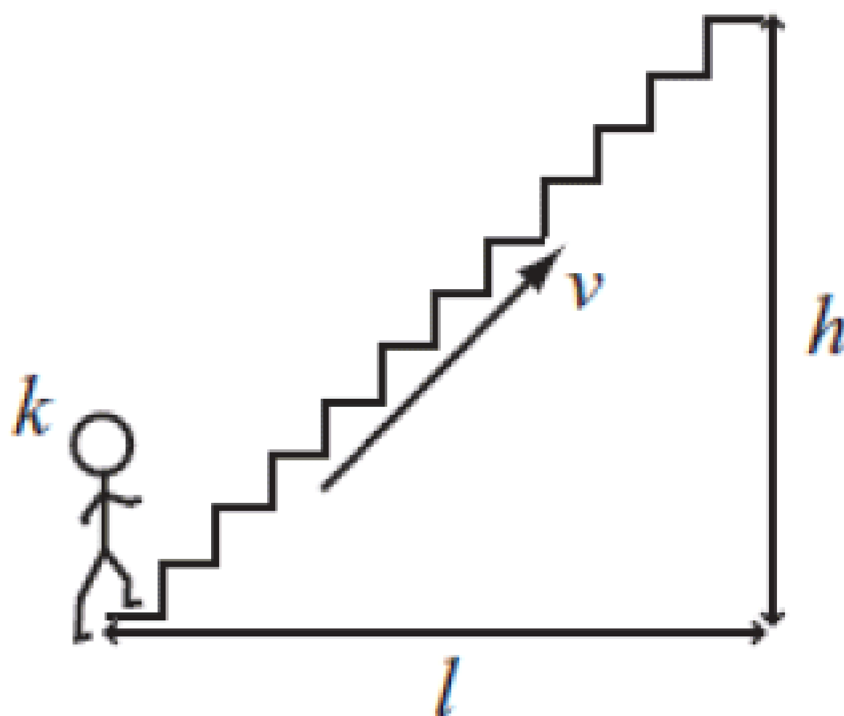


图 15.8　自动扶梯

图中，扶梯自身的长度为$\sqrt{l^2+h^2}$，此处可定义为 d，据此则可计算扶梯的重量。在$\frac{d}{v}$时间

内，即可到达扶梯的顶部。在该时间段内，$\frac{kd}{v}$千克将载入至扶梯中，即扶梯上乘客的整体质量。由于每秒上升$\frac{hv}{d}$距离，因而相应势能可表示为$\frac{kd}{v}\times g\times\frac{hv}{d}=kgh$。

需要注意的是，前述讨论并未涉及扶梯的自身质量，类似情况还包括动能。由于每秒加入的动能为$\frac{1}{2}kv^2$，因而全部功率为$\frac{1}{2}kv^2+kgh$，对应作用力为$\frac{1}{2}kv+\frac{kgh}{v}$。不难发现，扶梯的水平长度被消去，且仅与扶梯的垂直高度相关。据此，传送带可视为自动扶梯的特例，即高度为 0。

维修工程师常抱怨人们沿扶梯上行时对其造成的损害，此时，扶梯需要施加额外的作用力，这也意味着增加扶梯的功率。另外一方面，增加行进速度可减少于扶梯上的停滞时间，进而减少整体负载质量，该结果反映于力学公式的第二种形式中。虽然扶梯的功率增加，但作用力却并未如此。当$v>\sqrt{2gh}$时，作用力随着速度的增加而降低。

若乘客沿扶梯下行，则可通过相同方式进行计算。此时，扶梯功率有所降低。在某一临界速度内，作用力将减少；随后，所需作用力则有所上升。

15.3.2 火箭燃料

针对前述火箭示例，其处理过程类似于自动扶梯。当乘客离开扶梯时，其运载质量将明显下降，这一点与火箭基本一致。对于火箭而言，鉴于燃料消耗，因而飞行距离越大，其质量则越小。

假设火箭以恒定功率 P 消耗燃料，且消耗率为每秒 k 千克。若火箭的初始质量为 m_0，且运行时间为 t，试计算燃料耗尽时的运行速度。该问题涉及微分方程，总体而言较为复杂。其中，火箭推力源自发动机处的喷射燃料。若喷射速度以及燃料质量为已知，则可求解对应动量。类似于传送带，还可进一步确定功率和作用力。相反，若功率已知，则可计算燃料的喷射速度。在每秒中，k 千克燃料将生成能量 P，因而喷射速度为$\sqrt{\frac{2P}{k}}$。

火箭运动方程源自该速度，此处可将其称作 u。据此，若火箭的初始速度为 v_0，且受到恒定重力作用，其飞行速度可通过下式确定：

$$v_0+u\log\left(\frac{m_0}{m(t)}\right)-gt$$

上式适用于燃料消耗问题。对于某些特定信息，则可加入相关数据值，如下所示：

$$v=v_0+\sqrt{\frac{P}{2k}}\log\left(\frac{m_0}{m_0-kt}\right)-gt$$

当计算此刻火箭的飞行高度时，需要对火箭方程执行积分计算。

通常情况下，此类方程较为复杂且变化较快，因而因避免使用数值方案。换而言之，可简单地计算不同时刻火箭的位置。对此，相关函数假设火箭位于恒定的重力场中。与此对应的是，练习 15.1 将引入处于变化状态的重力场。

currentPosition()函数采用了前述讨论中所提及的参数，并返回当前位置数据，如下所示：

```
function getRocketPosition(currentPosition, currentSpeed,currentMass, massBurnRate,
                           fuelVelocity, gravity, time)
  set MassBurned to massBurnRate * time
  set speed to getRocketSpeed(currentSpeed, currentMass,massBurned, fuelVelocity,
                           gravity, time)
  return currentPosition + speed * time
end function
```

getRocketSpeed()函数针对上述函数进行了有效的补充，如下所示：

```
function  getRocketSpeed(currentSpeed,  currentMass,massBurned,  fuelVelocity,
                         gravity, time)
  return currentSpeed + fuelVelocity *log (currentMass/(currentMass-massBurned))
                     - gravity * time
end function
```

15.4　本 章 练 习

【练习 15.1】试调整 getRocketSpeed()函数和 getRocketPosition()函数，并对变化中的重力场加以处理。修订后函数应需要计算特定点处的、基于重力的加速度。除此之外，读者还可进一步尝试计算任意方向上的速度，而非仅限于垂直方向。

15.5　本 章 小 结

本章讨论了功率和功这两个概念，并展示了与作用力、能量以及动量相关的多个示例。在实际处理过程中，还介绍了与火箭运行相关的基础运算。

第 16 章将考察弹簧、弹性对象以及振荡体等较为常见的对象。

至此，读者应掌握如下内容：

- 张力的含义及其作用方式。
- 如何计算绳索连接对象的运动行为。
- 滑轮与作用力和能量之间的关系。
- 如何计算传送带或自动扶梯上对象的运动行为。
- 如何计算火箭的运动行为。

第 16 章　振 荡 现 象

本章包含如下内容：

- 概述。
- 弹簧。
- 简谐振动。
- 阻尼简谐振动。
- 复杂弹簧。
- 弹簧的运动计算。
- 波。

16.1　概　　述

第 15 章曾讨论了不可伸缩的绳索，本章则继续考察拉伸弹簧的行为方式。特别地，当粒子绑定于弹簧一端时，还将分析该粒子的运动方式。另外，一类较为特别的弹跳形式称作振荡，此类运动出现于多种场合。随后，本章还将定义相关函数，进而描述可伸缩的复杂弹簧。最后，还将对此类概念进行适当扩展以对波形进行处理，并进一步解释光线的某些特征。

16.2　弹　　簧

当谈及到弹簧时，读者脑海中可能会映象出第 15 章所讨论的、理想化的不可伸缩绳索。这里，前者定义为轻质绳索，且自然长度为 l（经拉伸后可大于该长度值）。当拉伸时，弹簧将产生张力并作用于两个方向上。此类计算适用于橡皮筋和弹簧，考虑到弹性在碰撞过程中具有特殊含义，因而本章仅采用“弹簧”这一名词。

16.2.1　拉伸弹簧所产生的作用力

关于弹簧中的张力，其计算过程较为简单。为了有效地描述弹簧的拉伸程度，弹簧通常包含弹性系数 k，据此，拉伸弹簧中的张力正比于拉伸长度。此处，拉伸长度等于当前长度与其静态长度之差，并反映于下列胡克定律中：

$$作用力=-k\times拉伸长度$$

式中的负号表明，根据传统规则，作用力通常以反向呈现，即指向非拉伸平衡状态。如图 16.1 所示，若某一粒子绑定于弹簧一端，则该粒子将受到沿弹簧长度的反向张力作用。

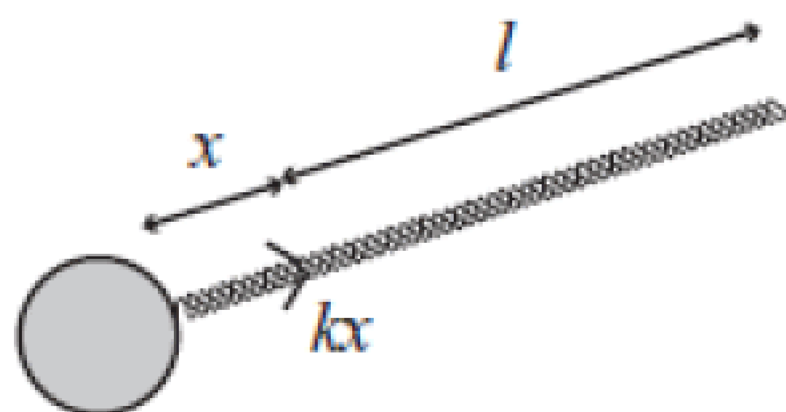


图 16.1　弹簧一端的粒子

某些弹簧仅可拉伸，也就是说，仅当向外拉伸该弹簧时，方可产生张力。此时，拉伸弹簧类似于刚性杆。另一个例子则是橡皮筋，在非拉伸状态，橡皮筋并不产生任何张力。除此之外，其他弹簧均具有压缩特征。若长度小于标准制度，则压缩将产生外向张力。综上所述，针对拉伸弹簧和可压缩弹簧，负长度变化将生成正张力。

当弹簧处于张力作用下（压缩或拉伸），则该弹簧蕴含势能，即弹性势能。拉伸弹簧时需要做功；当弹簧回弹时，则能量得到释放。对应能量如下所示：

$$能量=\frac{1}{2}\times k\times 拉伸长度^2$$

根据上式以及力学公式，可知弹性系数的单位为 kg s^{-2}。

需要注意的是，真实的弹簧一般无法实现无限度的压缩，并存在弹性极限。除此之外，若弹性系数显著增加，则弹簧将难以拉伸。同时，待到达弹性极限后，弹簧释放后将不再恢复其原始长度。作为一种极端情况，弹簧拉伸至极限点后，其构成分子之间的连接点可能会出现断裂情况，此时，弹簧很可能会折断。在当前环境中，若弹簧超出弹性极限，则可将其视为非扩展弹簧，且包含固定的内向张力。

16.2.2　通过弹簧测量重量

图 16.2 显示了弹簧的典型示例，其中，质量为 m 的对象绑定于弹簧一端，该弹簧的静止长度为 l。弹簧的另一端则固定于横梁或天花板上。

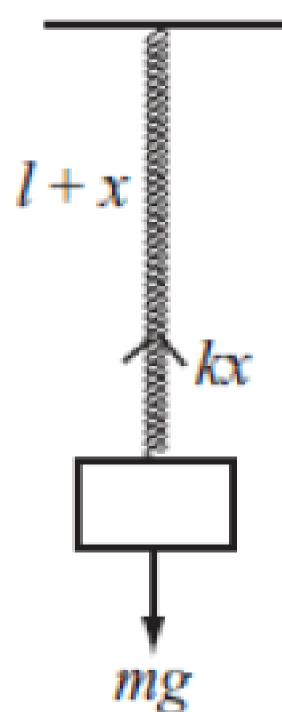


图 16.2　悬挂于弹簧上的粒子

若悬挂于弹簧上的对象处于平衡状态，则弹簧张力等于对象的重量，如下所示：

$$k\times 拉伸长度=mg$$

$$拉伸长度=\frac{mg}{k}$$

鉴于 g 和 k 值保持不变，因而弹簧的拉伸长度正比于悬挂于弹簧上的对象的质量。据此，可使用弹簧测量物体的重量。

若弹簧未处于平衡状态，且拉伸长度为 x，则作用力等于 $mg–kx$。二者之差包含了不同的情况，此处，对象在弹簧的作用下水平运动，该弹簧的未拉伸长度为 $l+\frac{mg}{k}$。该值十分有用，并可在计算对象运动时（该对象绑定于弹簧上）忽略重力作用。

16.3 简谐运动

当从平衡状态向下拉动并释放悬挂于弹簧上的某一物体时，该物体将上下跳动，这里，上下往复运动构成了振荡现象。振荡运动包含一类常见特征，有时也称作简谐运动或 SHM。

16.3.1 简谐运动方程

为了计算简谐运动方程，需要于先期考察胡克定律。通过牛顿第二定律，可得到如下算式：

$$ma = -kx$$

其中，x 表示为拉伸长度，m 表示为质量，a 表示为加速度。全部数据项均需要在同一方向予以计算。下列算式显示了另一个版本，即微分方程，如下所示：

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x$$

微分方程需要执行大量的计算，针对当前示例，简单函数即可胜任。sin()和 cos()函数的二阶导数为自身的相反数，如下所示：

$$\frac{d}{dt}\sin t = \cos t; \frac{d}{dt}\cos t = -\sin t$$

$$\frac{d^2}{dt^2}(\sin t) = -\sin t;$$

$$\frac{d^2}{dt^2}(\cos t) = -\cos t$$

对上述算式稍作调整即可得到通用方程，进而求解 SHM 的微分方程，如下所示：

$$x = A\sin(\omega t) + B\cos(\omega t)$$

其中，A 和 B 表示为任意常数，ω 表示为 $\sqrt{\frac{k}{m}}$。下列算式也表达了这一关系：

$$x = C\sin(\omega t + p)$$

其中，C 和 p 表示为任意常数。虽然两种形式均较为常见，但此处将采用后者。

【提示】方程间的不同形式可实现相应的调整。例如，下列方程

$$\sin(\omega t + p) = \sin(\omega t)\cos p + \cos(\omega t)\sin p$$

包含两个方程，如下所示：

$$A = C\cos p$$
$$B = C\sin p$$

当对 SHM 执行微分计算时，即可得到特定时刻的、绑定于弹簧一端的对象速度，如下所示：

$$\dot{x} = C\omega\cos(\omega t + p)$$

再次执行微分运算将得到如下算式：

$$\ddot{x} = -C\omega^2\sin(\omega t + p) = -\omega^2 x$$

对此，常可得到一组方程并包含多个有效解，且任意解均可表示弹簧上对象可能的运动方式。通过设置初始条件，即可选取对应解。换而言之，可选取对象在初始阶段的行为状态。例如，若对象被下拉距离 d 并于随后释放，则有 $C = d$，$p = 0$。若于平衡处上推对象，且初始速度为 v，则有 $C = \frac{v}{\omega}$，$p = \frac{\pi}{2}$。稍后将对此予以深入分析。

这里的问题是，如何描述具体的运动行为？例如，当 $t = \frac{p}{\omega}$，由于 sin(0) = 0 和 cos(0) = 1，则粒子位于 0 处且速度为 $C\omega$。若 C 为正值，则拉伸长度逐渐增加，直至 $t = \frac{1}{\omega}\left(\frac{\pi}{2} - p\right)$时，到达 C 的最大值，并于随后再次减至 0 且于相反方向返回。最终，在 $t = \frac{1}{\omega}(2\pi - p)$ 时返回 0 处。

如图 16.3 所示，对应行为体现了第 4 章所讨论的正弦波。其中，$\frac{2\pi}{\omega} = 2\pi\sqrt{\frac{m}{k}}$ 定义为运动周期，即一次完整振荡所需的时间；ω 值表示为频率；p 值则称作运动的相位，即波形沿时间轴移动的距离；C 则表示为振幅，即距离平衡位置的最大位移。

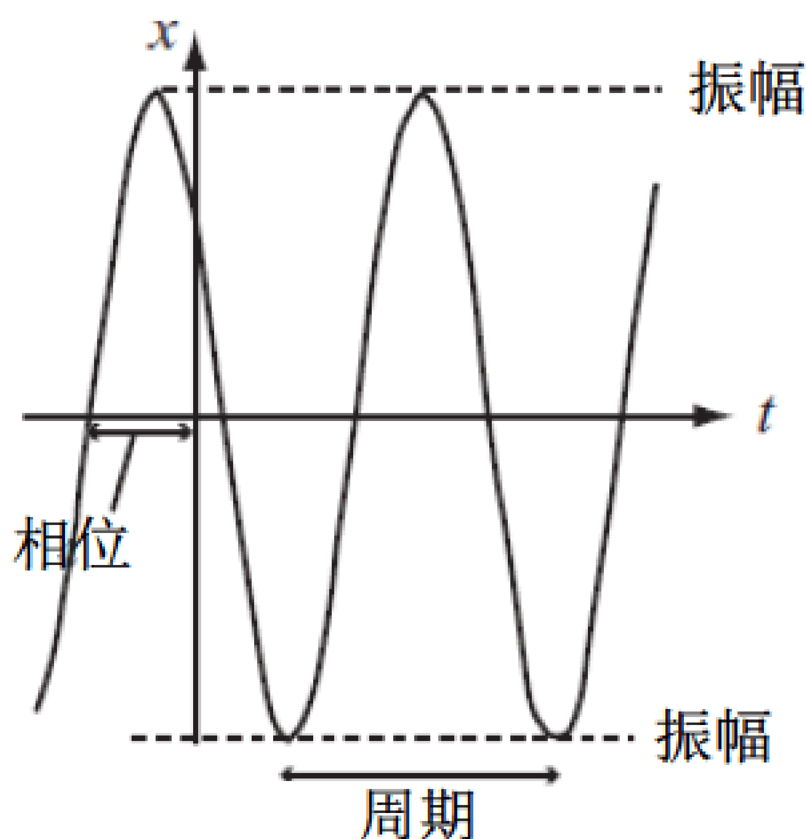


图 16.3　一段时间内 SHM 作用下的粒子位置

16.3.2　其他 SHM 示例

回忆一下，当粒子沿正弦波运动时，其位置变化等价于轮胎圆周上一点的 y 坐标，这也意味着，轮胎上的一点以 SHM 方式运动。实际上，图 16.4 显示了与此相关的 C 和 p 值。其中，C 表

示轮胎半径，p 表示行进距离。

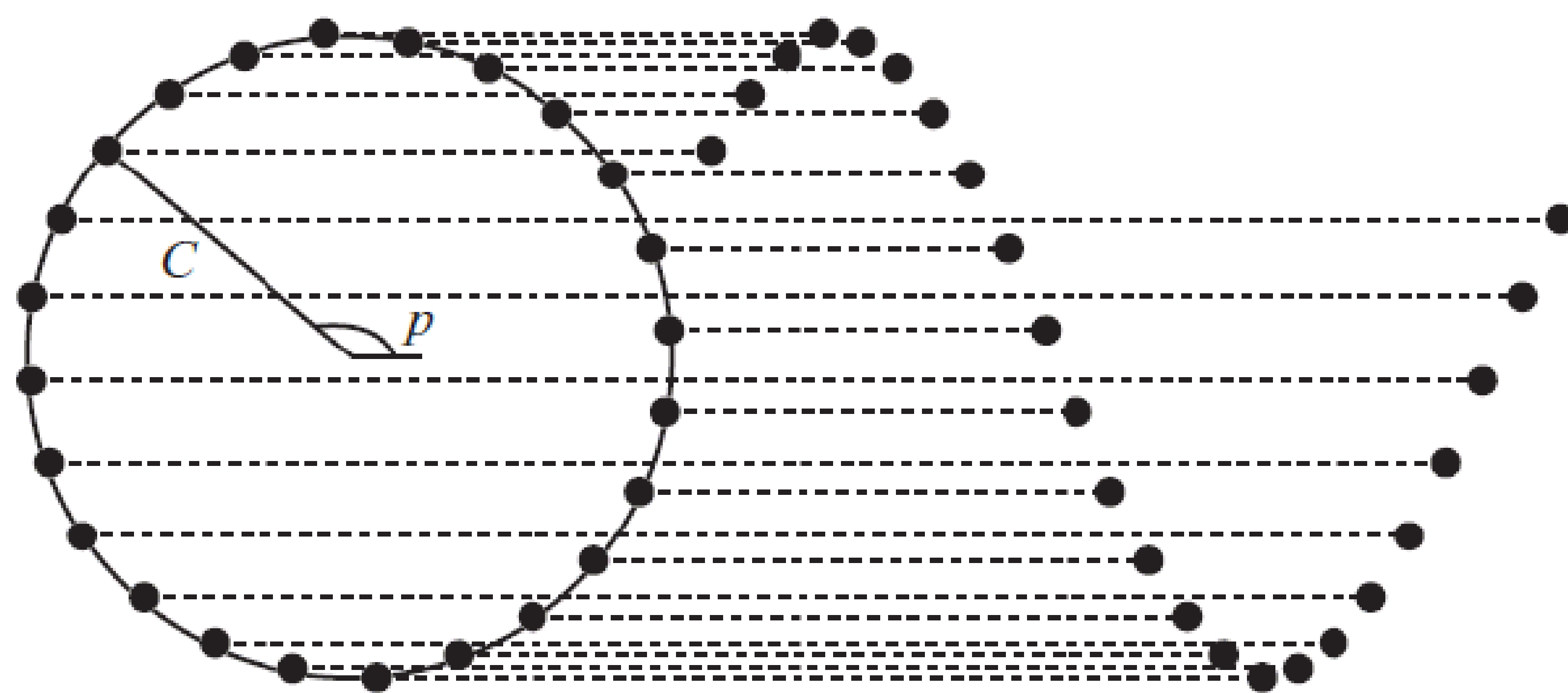


图 16.4　SHM 和圆周运动

此行为也体现了活塞的驱动方式。若将连杆与轮胎上的某一点连接，且轮胎转动时该连杆上下滑动，则连杆一端的运动行为类似于 SHM。

SHM 还出现于钟摆的运动中，再次强调，其运动状态仅是一类近似结果，且仅适用于摆动幅度较小的情况下。若对象通过轻质、不可伸缩的绳索悬挂于一端，当运动角度为 θ 时，该对象受到源自自身重量的作用力 W，以及来自绳索的张力 T。特别地，对象上的径向力（用以提供转矩）为 $-W\sin\theta$。其中，负号表明作用力与当前角度呈相反方向。如前所述，针对较小的 θ 值（振荡范围约为 5°），$\sin\theta$ 约等于 θ。因此，作用于粒子上的转矩约为 $-Wl\theta$。这体现了胡克定律的另一个示例，如下所示：

$$\text{转矩} \approx -Wl\theta \approx -mgl\theta$$

$$\text{角加速度} \approx \frac{-mgl\theta}{\text{转动惯量}} \frac{-mgl\theta}{ml^2} = \frac{g\theta}{l}$$

其中，粒子的质量被消去，且频率仅与绳索的长度有关。最终，周期长度的平方根呈正比。

其他 SHM 示例还包括波动水面上的浮标、吉他弦的振动、晶体中原子的振荡、交流电的变化以及一段时期内动物种群数量的变化。若在运动行为中存在平衡位置，且作用力使其不断恢复至平衡位置，则该过程即会存在 SHM。

另外，波形自身也包含了基于 SHM 运动的组成成分。例如，光波由振荡电场和磁场构成，任意位置处的场强在一段时间内以 SHM 方式变化。其中，各振荡引入与其相邻接振荡，经短暂延迟后其后将再次产生振荡，其运动过程类似于不断前行的正弦波，稍后将对此予以分析。

16.3.3　参数计算

下面再次返回至前述内容所讨论的参数 C 和 p。此处，C 表示为振幅，p 表示为运动相位。若弹簧上的某一质体以同一频率振荡，针对粒子的运动速度以及距平衡位置的距离，其振幅和相位将根据不同情况而发生变化。

一种最为简单的计算方案则是获取时刻 0 时的速度 v 和伸展长度 d，对此，可采用 SHM 的速度和位置公式作为联立方程，如下所示：

$$d = C\sin(p)$$

$$v = C\omega\cos(p)$$

因而有

$$\frac{d}{v}=\frac{\tan(p)}{\omega}$$

$$\omega^2 d^2+v^2=C^2\omega^2$$

最终结果如下所示：

$$p=a\tan\left(\frac{\omega d}{v}\right)$$

$$C=\frac{\sqrt{\omega^2 d^2+v^2}}{\omega}$$

通常情况下，可针对任意时刻 t 直线进行相同计算：基于 C 的计算过程保持不变，而与 p 相关的计算调整为下列形式：

$$p=a\tan\left(\frac{\omega d}{2v}\right)-\omega t$$

16.4　阻尼简谐运动

在真实生活中简谐运动并非如此简单，不难发现，真实的振荡行为难以维持较长时间，且会在一段时间内产生能量消耗，即阻尼。针对基于阻尼的真实运动行为，其计算过程并不复杂。

16.4.1　DHM 方程

对 SHM 方程稍作调整即可得到阻尼简谐运动方程，对此，可向微分方程中加入阻尼系数，该系数正比于速度而非加速度，如下所示：

$$\ddot{x}=-\omega^2 x-2D\dot{x}$$

【提示】本书采用 $2D$ 表示阻尼系数，并以此简化后续计算。通常情况下，可使用字母 b 表示阻尼系数。

相应地，微分方程求解过程的复杂度也有所提升，并涉及“试探解”的用法，进而生成特定的二次方程。此处暂且忽略某些细节内容，最终结果如下所示：

$$x=Ae^{-rt}$$

其中

$$r=-D\pm\sqrt{D^2-\omega^2}$$

取决于 D 和 ω 值，上述方程将生成不同的结果。若将变量 α 定义为 $D^2-\omega^2$，若 $\alpha>0$，则方程包含实解，最终将得到如下所示的方程系列：

$$x=A\,e^{(-D-\sqrt{\alpha})t}+B\,e^{(-D+\sqrt{\alpha})t}$$

若 $\alpha=0$，则有 $D=\omega$，此时运动行为类似于前述情形，仅仅出现单值 r，进而可得到相对简单的方程，如下所示：

$$x = (A + Bt)\ e^{-Dt} = (A + Bt)\ e^{-\omega t}$$

若 α<0，则方程不包含实根，最终结果则表示为复数。此处并不打算对复数做过多讨论，但读者依然可通过虚数 i（根据定义，该数字等于−1 的平方根）求解最终的微分方程。据此，则可推导出下列公式：

$$x = C\sin(\varphi t + p)\ e^{-Dt}$$

其中

$$\varphi = \sqrt{-a}$$

尽管缺少明显特征，上述公式依然类似于 SHN，这里存在两处主要差别。首先，该公式包含附加的指数项（式中显示为负值）。相应地，运动过程中的振幅将在一段时间内减少。其次，频率也发生了变化，D 值越大，则频率越小。这一过程持续进行，直至 D 达到临界阻尼值，即 $D = \omega$。该值上方处构成了第一类运动行为，对应频率实际上为 0。换而言之，该行为并非是振荡，对象的运动轨迹遵循指数曲线。总而言之，DHM 包含 3 种不同的行为区域，如下所示。

- 欠阻尼：这类似于 SHM，但振幅以指数方式递减——$\alpha < 0$。
- 无阻尼运动：仅出现单一 r 值——$\alpha = 0$。
- 过阻尼：指数递减且不包含振荡现象——$\alpha > 0$。

图 16.5 显示了不同阻尼形式的不同运动行为。

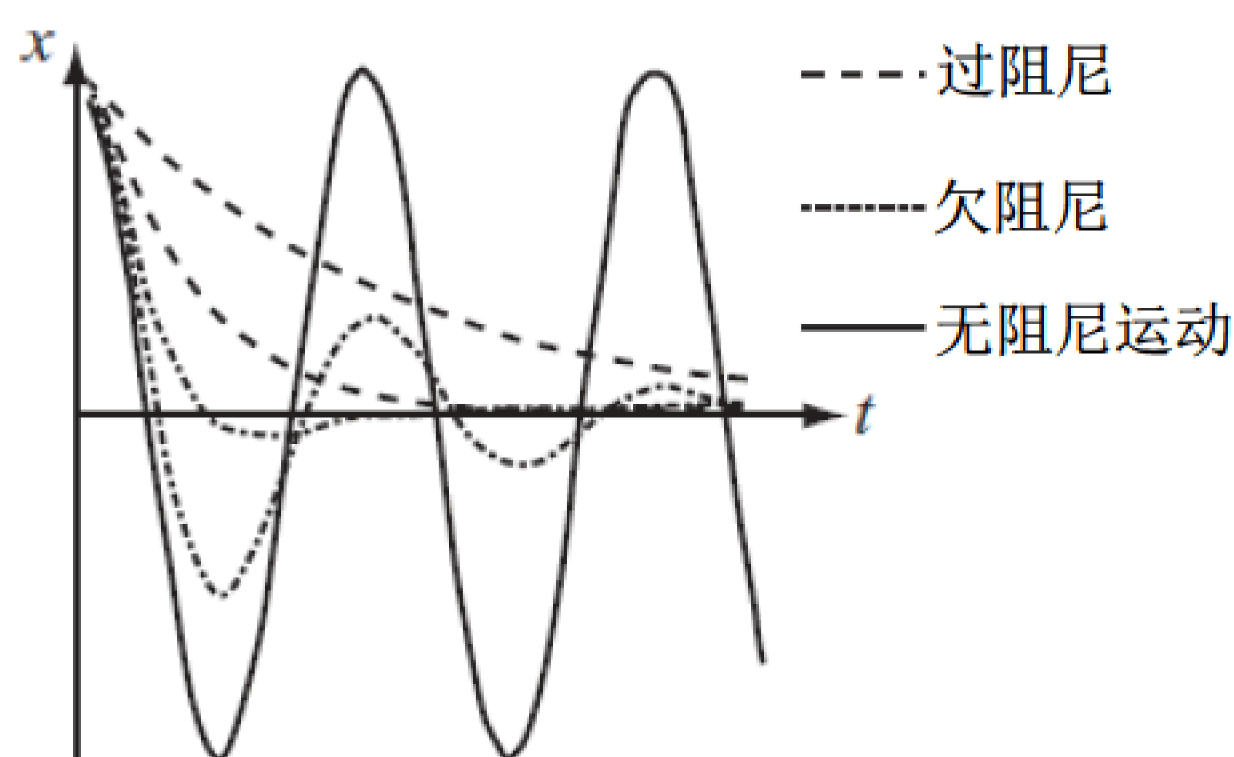


图 16.5　粒子基于不同阻尼级别的运动行为。全部运动均始于相同的初始条件，且全部弹簧均包含相同的弹性系数

16.4.2　实际阻尼计算

同样，可对各运动方程执行微分计算，进而获得速度函数。

欠阻尼可得到如下算式：

$$\begin{aligned}\dot{x} &= C\varphi\cos(\varphi t + p)e^{-Dt} - CD\sin(\varphi t + p)e^{-Dt} \\ &= C\varphi\cos(\varphi t + p)e^{-Dt} - Dx\end{aligned}$$

针对临界阻尼，最终结果如下所示：

$$\begin{aligned}\dot{x} &= Be^{-\omega t} - \omega(A + Bt)e^{-\omega t} \\ &= (B - \omega(A + Bt))e^{-\omega t} \\ &= Be^{-\omega t} - Dx\end{aligned}$$

回忆一下，针对临界阻尼，有 $D=\omega$。

对于过阻尼，最终结果如下所示：

$$\dot{x}=A(-D-\sqrt{\alpha})e^{(-D-\sqrt{\alpha})t}+(B-D+\sqrt{\alpha})^{e(-D+\sqrt{\alpha})t}$$

当采用更为简洁的方式表达时，过阻尼的微分结果如下所示：

$$\dot{x}=Ar_1e^{r1t}+Br_2e^{r2t}$$

这里，将相关数据值带入至上述方程中即可实现阻尼应用，但参数的计算过程则涉及较多细节内容，皆因速度参数更为复杂。

当与上述方程协同工作时，类似于弹性系数，D 也为已知常数，而参数 C 和 p 则表示为未知项。若已知时刻 t 处的距离和速度，则可通过方程组对其进行计算。

针对欠阻尼，初始阶段可得到下列算式：

$$d=C\sin(\omega t+p)e^{-Dt}$$

$$v=C\omega\cos(\omega t+p)e^{-Dt}-Dd$$

因而有：

$$p=a\tan\left(\frac{d\omega}{v+dD}\right)-\omega t$$

$$C=\frac{\sqrt{d^2\omega^2+(v+Dd)^2}}{\omega e^{-Dt}}$$

临界阻尼包含如下算式：

$$d=(A+Bt)e^{-\omega t}$$

$$v=(B-\omega(A+Bt))e^{-\omega t}$$

因而有：

$$\omega d+v=B(\omega t+1)e^{-\omega t}$$

$$(1-\omega t)d-tv=Ae^{-\omega t}$$

若 t=0，则临界阻尼将变得更加简单，如下所示：

$$A=d$$

$$B=\omega d+v$$

对于过阻尼，则有如下算式：

$$d=Ae^{r1t}+Be^{r2t}$$

$$v=Ar_1e^{r1t}+Br_2e^{r2t}$$

因而有：

$$r_1d-v=B(r_1-r_2)e^{r2t}=-2B\sqrt{\alpha}e^{r2t}$$

$$r_2d-v=-A(r_1-r_2)e^{r1t}=2A\sqrt{\alpha}e^{r1t}$$

这将生成如下算式：

$$A=\frac{r_2d-v}{2\alpha}$$

$$B=\frac{r_1d-v}{2\alpha}$$

16.5 弹簧的复杂性

尽管前述各节对弹簧的物理现象进行了综合讨论，但相关方程的某些重要结论依然值得关注，例如共振现象，共振可导致结构处于不稳定现象。除此之外，联轴弹簧也十分重要，其中，弹簧的物理行为将变得十分复杂。

16.5.1 共振与秋千

声音可振碎玻璃，相信大多数人对此均有所耳闻。除了高音之外，SHM 则是隐藏其后的主要原因。

此处对图 16.2 稍作改动，其中，弹簧上方绑定于振动杆上，且振动杆以驱动频率 f、振幅 A 上下运动，振动过程向当前系统中加入了作用力。当连杆向上运动时，这将增加源自弹簧的张力；当连杆下降时，则释放作用力。

尽管对象以不规则方式跳动，但依然会呈现出某种运动模式。特别地，若调整驱动频率，且在接近于弹簧的原有频率的过程中，对象的运动振幅将有所增加。当到达固有频率时，粒子的运动行为最为剧烈。实际上，从理论角度上讲，粒子的弹跳高度趋于无穷大。随后，若进一步增加驱动频率，则粒子运动行为趋于缓和。

运动振幅 C 和驱动频率之间存在简单的关系，如下所示：

$$C = \frac{kA}{\sqrt{m^2(f^2-\omega^2)^2+4D^2\omega^2}}$$

这里，kA 值表示驱动振荡所施加的最大作用力，即驱动力。

【提示】上述公式适用于驱动振荡为正弦曲线。顾名思义，正弦波具有正弦波形，通常表示为与 SHM 相关的波形。若驱动振荡包含其他模式，则该公式也随之产生变化，但这也仅限于常数因子，基本的运动行为仍保持一致。

当驱动频率等于固有频率 ω 时，根号下的第一项将被消除。在 SHM 中，当 $D=0$ 时，这意味着 C 将趋于无穷大；否则，C 反比于阻尼系数。最终，若系统中包含较小阻尼，则振荡行为将失去控制并呈现为螺旋形状态，即共振，而固有频率则称作共振频率。

这一原理亦体现于秋千中，若在秋千的最高点处施加作用力，并与振荡频率保持一致，最终，振荡幅度不断增加。又如，玻璃平面同样包含固有频率，若以同频率的声波“撞击”玻璃平面，该平面将会产生剧烈的抖动甚至破损。

综上所述，当在模拟环境中构建弹簧系统时，应设置相应的阻尼因子，或至少定义一个弹性极限条件。否则，弹簧系统可能会失去应有控制。

16.5.2 联接弹簧：链接运动

两个弹簧彼此整合后将形成联接（耦合）弹簧，其中，各弹簧的运动行为将受到另一个弹簧

的影响。同时，能量将在两个弹簧之间进行传递。最终，联接弹簧可导致极为复杂的运动行为。这里并不打算讨论其数学内容，但会考察一类特殊的运动结果。

若两个相同的联接弹簧设置为同相位运动，则二者将以固有频率平行摆动，尽管它们受到源自联接弹簧的张力。若弹簧设置为异相运动，则在弹簧一下坠过程中，弹簧二处于上升阶段；随后，二者在同一时刻释放，且持续进行异相振动。在此过程中，其运动频率则有所提升。这里，该过程称作运动的两个固有模式。

总体而言，任意两个联接振动均会生成基于两个固有模式的系统，且各模式包含自身的频率。同时，相关模式取决于两个振子和联接弹簧的频率。系统的其他运动可视为两个振动的线性和。在声学中，此类概念十分重要。

16.6　弹簧运动的计算过程

根据前述数学和物理内容，本节将讨论 3 个函数，进而计算绑定于任意弹簧上的粒子的运动行为。同时，弹簧将被赋予多种特征，例如弹性系数和长度。除此之外，弹簧上的对象包含一定的质量以及重力。

16.6.1　基于弹簧的作用力

上述 3 个函数分别适用于不同的场合，其中，第一个函数可定义为纯数学系统。据此，该函数返回弹簧上对象的作用力。此时，需要单独使用重力。在大多数场合，这可视为唯一可行方案，当弹簧端点未固定时尤是如此。forceDueToSpring()函数如下所示：

```
function forceDueToSpring(end1, end2, velocity1,velocity2, springLength, elasticity,
                              damping, elasticLimit,compressiveness, minLength)
  //The object you're interested in is attached to end2
  set v to end1-end2
  set d to magnitude(v)
  if d=0 then return vector(0,0)
  //skip for this time-step if they coincide

  //loose elastics have no force when compressed
  if d<=springLength then
    if compressiveness="loose" then return vector(0,0)
  end if

  //apply second elastic limit (inextensible behavior)
  if d>=elasticLimit*1.2 or d<=minLength*0.9 or (d<=springLength*0.9 and
                               compressiveness="rigid") then
    return "bounce"
  end if

  //apply first elastic limit (increased force and damping)
```

```
  if d>=elasticLimit or d<=minLength or (d<=springLengthand compressiveness=#rigid)
                                                then
    multiply elasticity by 20
    set damping to max(damping*10,20)
  end if

  //calculate force by Hooke's law
  set e to d-springLength
  set v to v/d
  if damping>0 then
    set vel to component(velocity1-velocity2,v)
    set f to damping*vel+elasticity*e
  else
    set f to elasticity*e
  end if
  return f*v
end function
```

forceDueToSpring()函数中的唯一复杂之处在于处理弹性极限条件。由于涉及某些非真实环境，因而与真实弹簧相比，其模拟过程有时更为复杂，例如弹簧超出了自身的弹性极限。当然，此类情况可得到有效的避免。除此之外，其他情况还包括误差积累所造成的共振现象。

对此，可构建一类分层式弹性极限系统，除了设定弹性极限条件之外，还可适当增加弹性系数和阻尼系数。增加弹性系数的直接效果是生成较大的内向作用力，这一点十分重要。由于阻尼系数可确保系统迅速释放能量，因而阻尼系数十分必要。这也意味着，在下一次振荡过程中，运动行为不会超出弹性极限条件。

另外，还可定义第二个弹性极限条件，约为第一个条件的 1.2 倍。若弹簧试图超出这一标准，则可将该过程视为与弹簧垂直的、实体壁面之间的碰撞行为，进而保证弹簧不会超出第二极限条件。

【提示】若弹簧超出了极限条件，其自身物理属性将发生显著改变。例如，静态长度增加，或者弹簧自身被折断。

16.6.2 非阻尼和非联接弹簧

第二种方法则相对简单，其中涉及非阻尼弹簧和非联接弹簧。当采用此类弹簧时，弹簧将相关对象绑定于某一固定点，该对象可在各方向上自由运动。例如，读者可单击对象或抛掷对象；或者，该对象也可作为碰撞系统中的一部分内容。此时，读者可使用能力守恒定律处理误差问题。在各时间步内，当处理独立作用力时往往会产生此类误差。鉴于粒子的全部能量表示为常量，若位置已知，则可计算任意时刻的速度。实际上，该系统甚至可处理（人为控制下的）“固定”点处于运动状态下的模拟环境。particleOnSpring()描述了非阻尼弹簧和非联接弹簧的工作状态，如下所示：

```
function particleOnSpring (end1, end2, speed,direction, mass, totalEnergy,
                          springLength, elasticity,compressive, timeStep, g)
```

```
//Returns a list of position, speed, direction, and total energy.
//totalEnergy can have the value "unknown",
//in which case the function calculates it and returns it.

set v to end1-end2
set d to mag(v)
set e to d-springLength

if totalEnergy="unknown" then
  //calculate energy
  set totalEnergy to mass*speed*speed/2
  if e>0 or compressive=TRUE then
    set epe to elasticity*e*e/2
    add epe to totalEnergy
  end if
  if g>0 then
    set gpe to mass*g*end2[2]
    subtract gpe from totalEnergy
  end if
end if

//calculate force
set f to vector(0,mass*g)
if e>0 or compressive=TRUE then
  if d>0 then
    add v*elasticity*e/d to f
  end if
end if

//calculate new position
set a to f/mass
set displacement to direction*speed*timeStep + a*timeStep*timeStep/2
set pos to end2 + displacement

//calculate new elastic energy
set newd to mag(pos-end1)
set newe to newd-springLength
if newe>0 or compressive=TRUE then
  set epe to elasticity*newe*newe/2
otherwise
  set epe to 0
end if

//calculate new kinetic energy and hence speed
set ke to totalEnergy-epe+mass*g*pos[2]
if ke<=0 then //NB: for safety
  set speed to 0
otherwise
  set speed to sqrt(2*ke/mass)
  set velocity to norm(displacement)
end if
```

```
  return Array(pos,speed,velocity,totalEnergy)
end function
```

16.6.3 纯 DHM 振荡

最后一个函数最为简单，该函数表示为纯阻尼简谐运动（若阻尼为 0，则为 SHM）。若输入初始位置、速度以及时刻 t，则函数负责计算于该时刻处的位置和速度。其中，函数通过 3 个辅助函数执行计算，且主函数为 calculateDHMparameters()。由于对此计算同一参数可视为一类冗余计算，因而最佳方案则是存储对应数据。这里，第一个函数计算参数以及运动形式，其他两个函数则计算基于该结果的实际值。calculateDHMparameters()函数如下所示：

```
function calculateDHMparameters(initialPos, initialVel,elasticity, damping)
  set omega to sqrt(elasticity)
  set d to damping/2
  set alpha to elasticity-d*d
  if d=0 then
    set p to atan(omega*initialPos/initialVel)
    set c to sqrt(elasticity * initialPos *initialPos + initialVel * initialVel)/
               omega
    return array("SHM", p, c)
  else if d<omega then
    set v to initialVel + d * initialPos
    set p to atan(initialPos * omega / v)
    set s to initialPos * initialPos * elasticity + v * v
    set c to sqrt(s)/omega
    return array("UnderDamped", p, c, sqrt(-alpha))
  else if d=omega then
  return array("Critical", initialPos, omega *initialPos + initialVel)

  else
    set sq to sqrt(alpha)
    set r1 to -d-sq
    set r2 to -d+sq
    set a to (r2*initialPos - initialVel)/(2*sq)
    set b to -(r1*initialPos - initialVel)/(2*sq)
    return array("OverDamped", a, b, r1, r2)
  end if
end function
```

待 calculateDHMparameters()函数定义完毕后，则可得到 OscillatorPosition()函数，其定义如下所示：

```
function getOscillatorPosition(elasticity, damping, params, time)
  set omega to sqrt(elasticity)
  set d to damping/2

  if params[1] is
```

```
    "SHM": return params[3] * sin(omega * time + params[2])
    "UnderDamped": return params[3] *
     sin(params[4] * time + params[2]) * exp(-d * time)
    "Critical": return (params[2] + time * params[3]) * exp(-d*time)
    "OverDamped": return params[2] * exp(params[4]*time +params[3] * exp(params[5]*
                                         time)
  end if
end function
```

待 calculateDHMparameters()和 getOscillatorPosition()函数调用完毕后，可于随后调用 getOscillatorSpeed()函数，其定义如下所示：

```
function getOscillatorSpeed(elasticity, damping, params, time, pos)
  //determine pos before running this function
  set omega to sqrt(elasticity)
  set d to damping/2

  if params[1] is
    "SHM": return params[3] * omega * cos(omega *time + params[2])
    "UnderDamped": return params[3] * omega * cos(params[4] *time + params[2]) *
                                                 exp(-d * time) - d*pos
    "Critical": return params[3] * exp(-d*time) - d*pos
    "OverDamped": return params[2] * params[4] *exp(params[4]*time] + params[3]
                                      *params[5] * exp(params[5]*time)
  end if
end function
```

上述 3 个函数稍复杂于前述 SHM 和 DHM 方程。需要注意的是，函数间传递的变量均定义为全局变量，因而 3 个函数相互依赖。

16.7　波

当联接（耦合）振子彼此连接时，某一端的振动可向邻接振子传递能量，因而全部能量沿当前系统逐次传递，这一点与牛顿摆十分类似。这里，振动的“串联”方式称作波。

16.7.1　波运动

波可视为一组独立的耦合振子，各振子均处于振动状态，并沿直线进行传播。其中，当前振动与前次振动为异相关系。

如图 16.6 所示，该图所示轮廓为正弦波。当然，波形并非总是正弦模式，但无论何种形状，各振荡行为均可视为某一端波振荡的“副本”。

振动驱动形成的波体现了第二种描述方式，根据此观点，该波称作运动波形。这里，波形可视为以某种速度运动的对象，这里，将波形描述为对象并不准确。作为一种虚拟对象，可将其视

为传输于某种介质的能量包。



图 16.6　一系列的联接振荡

当从波形视角加以讨论时，波速表示为一段时间内波形的移动距离，该值由介质的物理属性确定。对于正弦波，可计算一段时间内波前（wavefront）的位置。

待波速 v 及其频率计算完毕后，则可计算连续波前之间的距离，该值也称作波长且记为 λ。据此，3 个数据项之间的关系可表示为 $v = f\lambda$。

16.7.2　波类型

波存在两种主要形式，即横波和纵波，图 16.6 即显示了横波示例。对于横波，对应振子垂直于波的运动方向，例如水波和电磁波。其中，光线也可视为一种电磁波。

纵波通常难以描述，螺旋弹簧则是其中一例（多出现于儿童玩具中，例如弹簧狗）。当弹簧拉伸并向另一端猛然收缩时，涟波沿线圈移动，各线圈沿涟波的有运动方向往复振动。

声波也可视为波的一种类型，并由空气分子往复振动形成，这将产生较小的低压和高压区域，二者趋于恢复至平衡状态。其中，低压区域处于稀疏状态，而高压区域则处于压缩状态，如图 16.7 所示。

虽然涉及不同的物理原因，但横波和纵波从本质上讲具有相同的行为方式。例如，可通过压力-时间图显示纵波。另外，类似于横波，纵波也包含反射、折射以及漫反射现象。

如图 16.8 所示，两种波形可采用相同的平面图予以绘制，其中，波采用基于特定波前的一系列的直线表示。

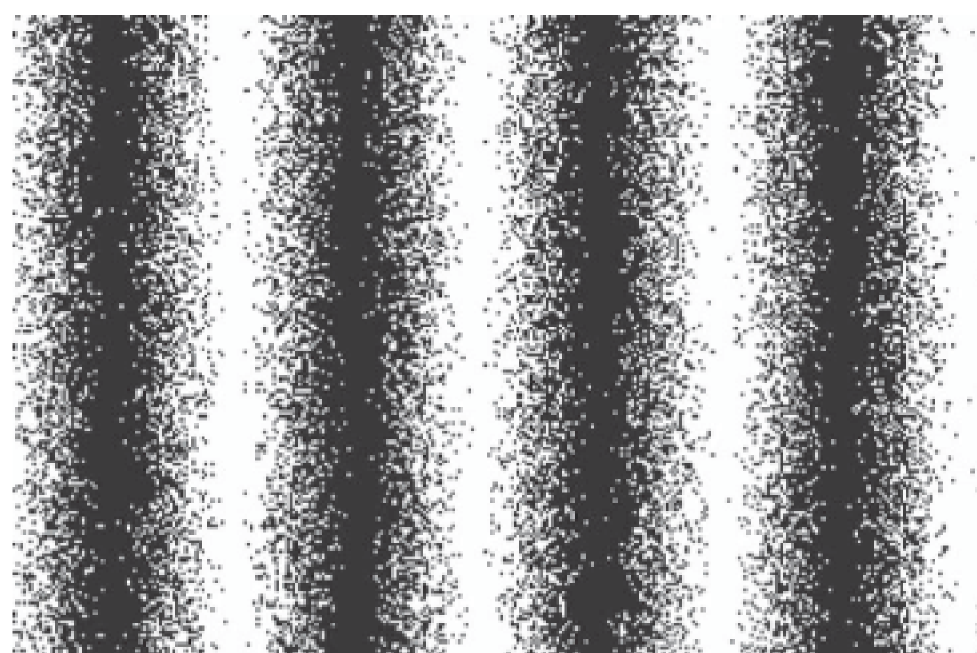


图 16.7　纵波

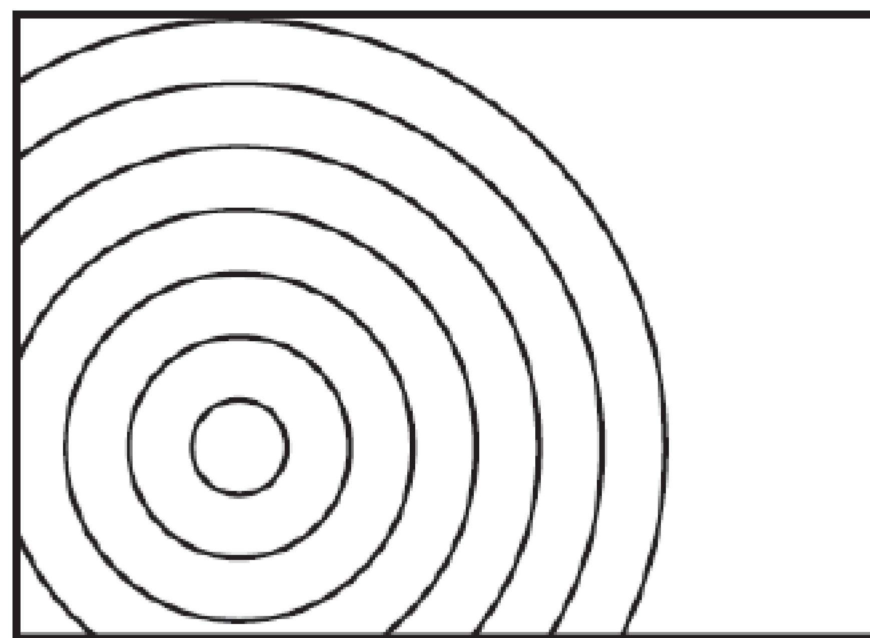


图 16.8　波采用连续波前加以表示

16.7.3　波的叠加和削减

由于波被视为虚拟对象，因而多个波可穿越同一介质。实际上，该现象亦有章可循。例如，

空气中包含了多种不同类型的光波。在现实生活中，全部行为均可视为空间中电磁场的波动，此时，能量传输则显得相对次要。

鉴于波包含重要的属性，因而读者可从两方面对其进行考察。波之间可合成为某一更为复杂的波，例如，可通过叠加各点处的位移值实现两个波的叠加，如图 16.9 所示。

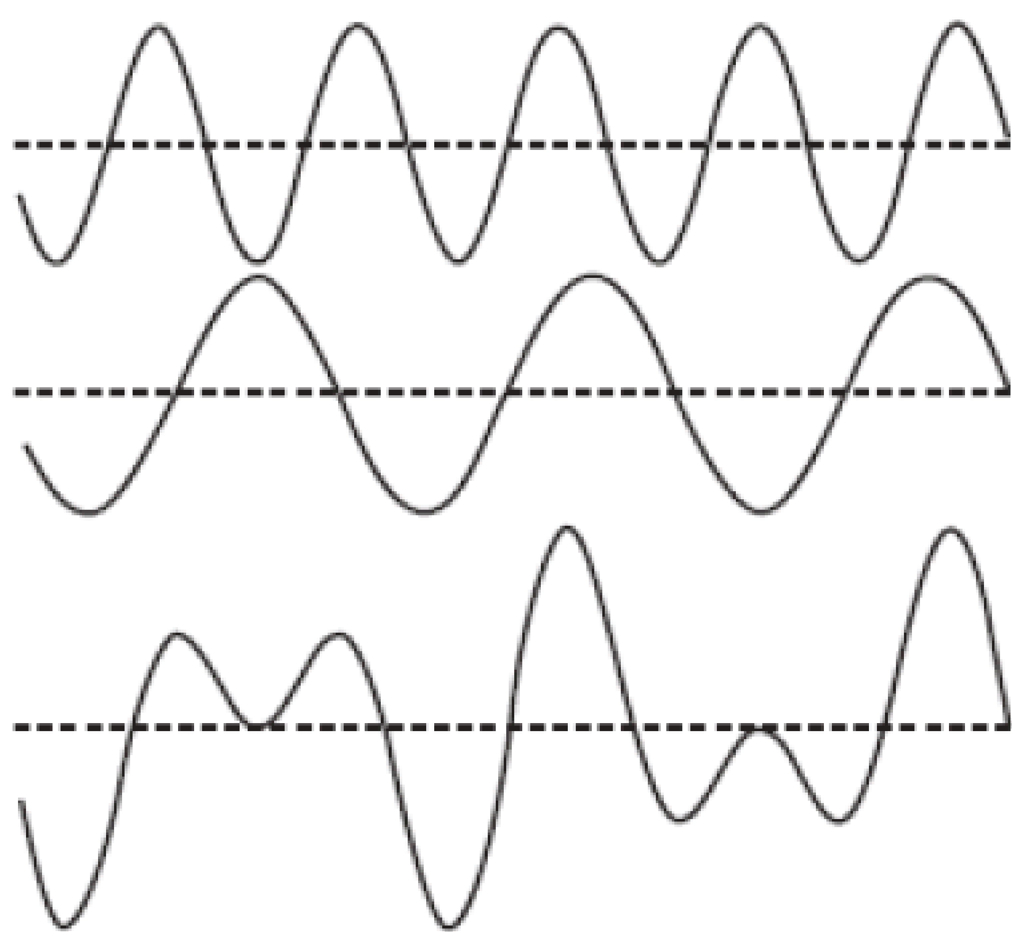


图 16.9 波的叠加

波经过合成后可尝试某些较为有趣的效果。具体而言，若叠加两个异相波，则最终结果将变为一条直线；若 3 个波彼此异相 2π/3，则叠加后将生成同一结果。据此，为了减少电流，发电厂常采用异相波进行电力传输。

相对于波的合成，还可对波进行分解。例如，一种称作傅里叶分析的技术可对任意波形进行分解，无论该波形如何复制，最终均可得到一类振幅、相位变化的正弦波。实际上，当听取声音时，人们即在分离对应于不同声源的各类波形。

波形分解为单一正弦波称作光谱分析，波形的光谱可视为波反射器的“签名”，且常用于各种分析环境中。例如，光谱分析可用于区分不同种类的化学元素。当某物质燃烧时，各元素发射光辐射的特征光谱。通过这一手段，可对恒星的化学成分予以分析，进而获得其他有用信息。

相比于光线，乐器同样包含特征声谱（即音色），并由主音的整数和小数倍数构成。其中，声音根据基于频率的特定音符被人们所感知。独立声源、单频率的复杂波形感知可视为一类心理计算，这一点基本等同于单色复杂光谱的感知过程，第 20 章还将对此加以分析。

16.7.4 波的物理行为

与波相关的重要行为实际上可视为其物理结果，下面首先对反射现象予以介绍。当波与障碍物碰撞时，取决于波以及障碍物的自身特征，波相对于障碍物产生反射现象。若障碍物不吸收能量，则波按照原有路径返回。如图 16.10 所示，波的交互方式类似于弹性碰撞，并以相同角度反向弹回。波与墙面之间的碰撞角度称作入射角，而反弹角度则称作反射角。

波的另一种行为是折射。当波速发生变化时，即会产生折射现象。假设车队以每小时 60 英里的速度行进，且前方区域使其需以每小时 30 英里速度行进，则在低速区域内，车队中的车辆将彼此聚集。同样，波前也会产生类似的情况。当波穿越某一介质且速度变慢时，波长也将随之降低，该结论可直接从波形方程中得到。另外，由于波的频率仅与振动的驱动过程相关，因而对

于某一既定波形而言，其频率保持不变。

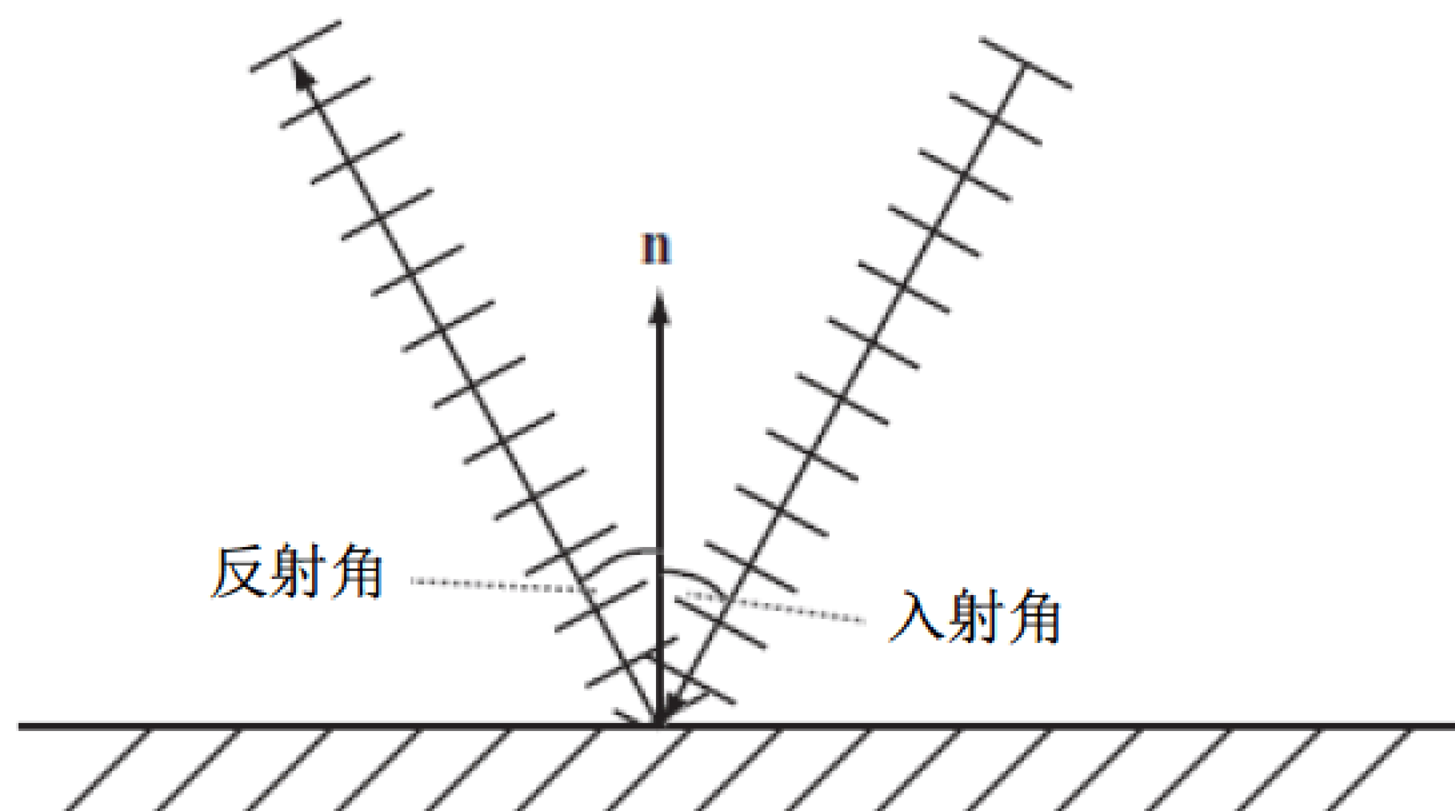


图 16.10　波的反射

改变波长可能会引发其他结果，如图 16.11 所示。当连续波前以某一角度穿越新介质时，波的某一部分将与该介质的界面产生碰撞，这将改变波的传输方向，类似于拖拉机或坦克转向时调整履带的相对速度。

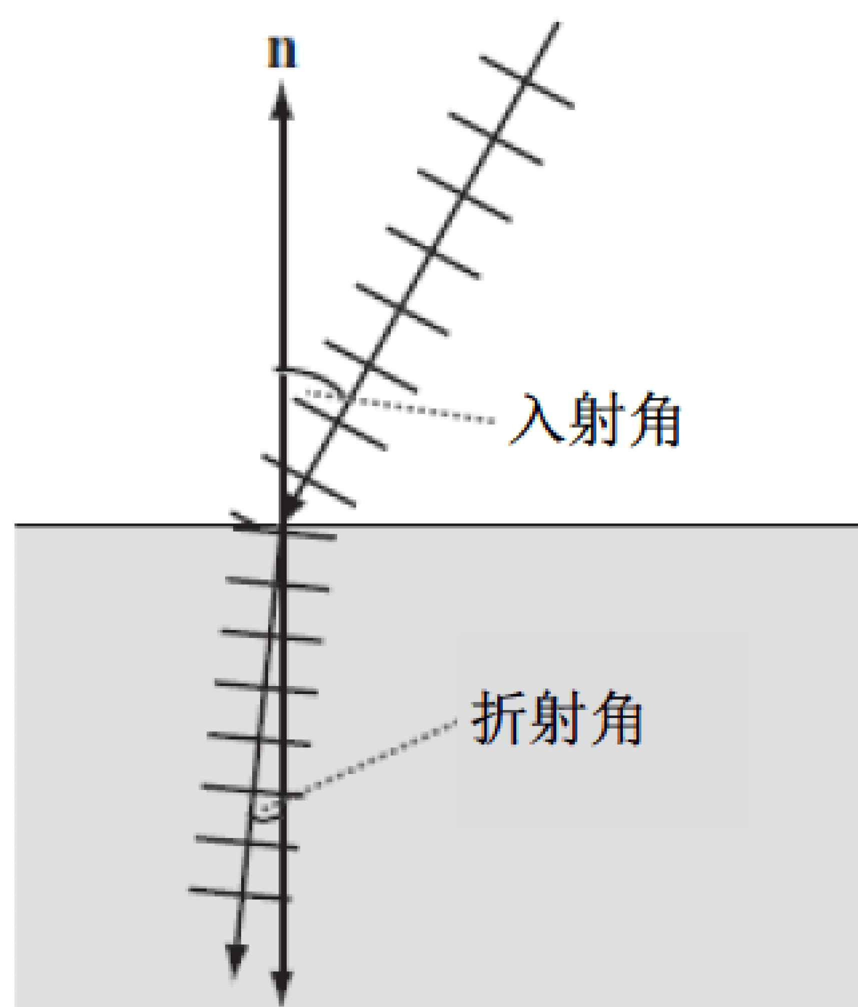


图 16.11　波的折射

除此之外，当吸管插入水中时也会出现类似的现象。在进入水面处，吸管呈现为弯曲状态。若不同的波长以不同的速度在水中行进时，则吸管将呈现为更为复杂的外观。最终，波不同程度地改变其传输方向。多种不同波长形成的波可分为一个光线扇形，并可形成三棱镜效果，例如彩虹。这里，波的偏移量可通过斯涅耳定律加以描述，该定律表明，若 α 和 β 分别表示为入射角和折射角，则二者关系如下所示：

$$n_1\sin\alpha = n_2\sin\beta$$

其中，n 表示介质的固定属性，即折射率。对于光线而言，该值等于光线在真空中的速度 c 与当前介质中波长速度之间的比值。

下面将要解释的一种现象称作多普勒效应。对此，假设 Heathcliff 和 Cathy 乘坐火车相向而行，当 Heathcliff 向对方呼喊时，相对于 Cathy，火车的运行速度导致声音波长变短。当 Heathcliff 前行时，各波前距其前一波前的距离缩短。

除此之外，还可计算多普勒频移。此处，多普勒频移表示波长据此移动的距离。若火车的行

驶速度为 s，声音频率为 f，则波长将减少 $\frac{s}{f}$。如果该过程使得波长小于 0，则问题也会随之而来，例如，s 等于声音速度（约为 330m/s）时，声波将被发射源超过，这将导致声爆的出现。Dang Heathcliff 消失于夕阳中，上述计算同样适用，此时可采用 s 的逆置速度。若 Cathy 位于山顶某处（仍位于车厢内），则可通过点积运算计算其方向上的火车速度。

多普勒效应还可作为宇宙大爆炸的主要证据之一，如前所述，元素的特征光谱可用于确定恒星的组成部分。当检测某一恒星的光谱时，对应结果基本处于红移状态。换言之，由于星球不断远离，因而其波长减少。而且，恒星的运动频率取决于其与观察者之间的距离，因而这描述了宇宙自爆炸以来的持续膨胀过程。

【**提示**】这里，可将地球或太阳系置于宇宙中心位置。全部恒星彼此远离，且宇宙自身处于膨胀状态。

衍射现象同样值得关注。当波与部分障碍物碰撞时，例如包含空洞的墙面，则会产生衍射现象。如图 16.12 所示，其中，障碍物类似于一个新的波源。

当两个空洞或缝隙彼此靠近时，其结果较为有趣。由于两个波具有相同的波长、频率和速度，在各点合成后将生成干涉图样。在图 16.13 中，位于不同点的两个波彼此异相且无能量传播；而在其他点处，波之间处于同相状态，进而生成双重效果。此时，若将一组探测器以直线方式放置，则会查看到干涉条纹。该现象也可在光波中出现，这也是将光线视为波的主要证据之一。

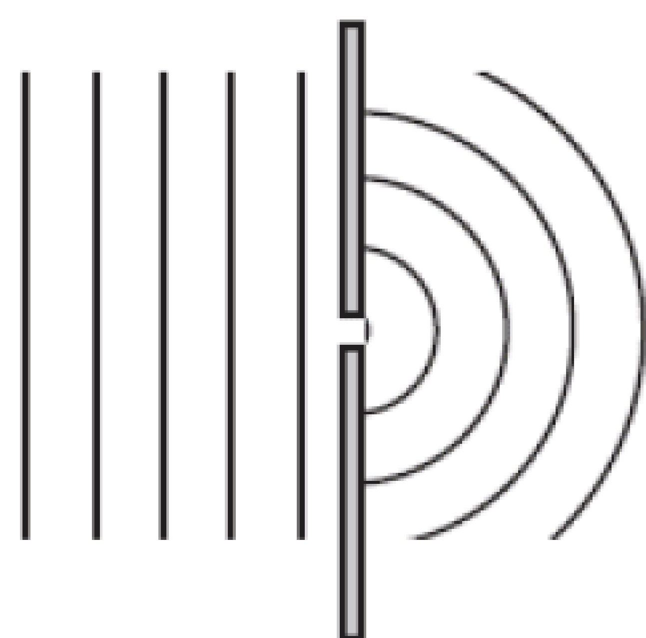


图 16.12　波的衍射

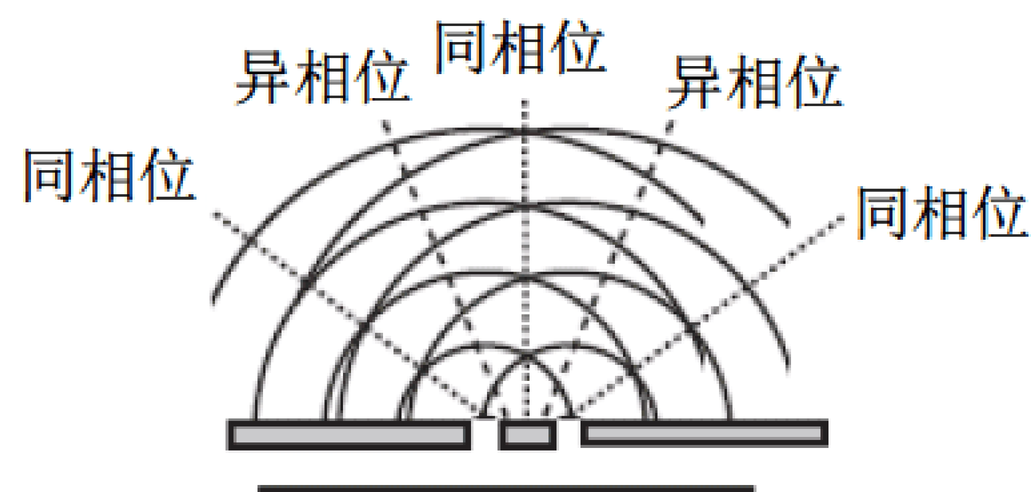


图 16.13　双缝衍射中的干涉图样

16.8　本 章 练 习

【练习 16.1】尝试将本章所讨论的函数整合至某一系统中，进而可实现基于虚拟弹簧的对象的单击、拖曳以及抛掷。在整合过程中，读者将会发现函数之间存在细微的差别，其中，较难的函数则是 forceDueToSpring()。

16.9　本 章 小 结

本章讨论了绑定于橡皮筋或弹簧上的对象的运动方式。鉴于弹簧可连接对象，制作虚拟布料、

绳索链条以及类似的连接对象系统，因而可生成多种物理模拟环境。另外，读者还可通过波了解连接对象与虚拟能量包之间的关联数量。

第 17 章将继续讨论数学运动，并将其扩展至三维空间内。

至此，读者应掌握如下内容：

- 弹簧的工作方式。
- 如何计算拉伸或压缩弹簧的张力和能量。
- 术语“弹性系数”、“阻尼”以及“弹性极限条件”的具体含义。
- 如何计算基于简谐运动或阻尼谐运动的粒子的位置和速度。
- 共振和耦合联接现象。
- 波、频率、波速的含义。
- 反射、折射、漫反射以及多普勒效应的构建方式。

第 4 部分　3D 数学

前述章节讨论了二维与三维环境中的大部分内容，其中，三维中的向量、作用力、能量以及动量与二维环境基本相同。尽管碰撞过程更为复杂，但基本技术并无太多变化。本书第 4 部分将数学和物理内容引入至三维环境中。

本部分内容首先介绍三维空间，及其在二维屏幕上的表达方式。随后，将对向量实施进一步扩展，并引入“转换”这一概念。第 19 章将处理 3D 形状的碰撞检测问题，第 20 章将考察光照和着色机制，第 21 章将探讨各类建模技术，进而构建复杂对象和运动表面，例如水波。

与本书中的某些通用技术相比，三维环境涉及了较为广泛的内容，且多数内容均会在本书中予以简要介绍。另外，针对完整的数学知识以及相关话题所涉及的其他信息，读者可参考本书附录部分以获取更为详细的内容。

第 17 章　3D 形状

本章包含如下内容：

- 概述。
- 3D 向量。
- 渲染技术。
- 光线投射。

17.1　概　　述

本书前述内容所讨论的大多数问题仅适用于 2D 环境，尽管添加一个数字即可实现增加一个维度，但具体过程还将涉及诸多内容。首先，计算机屏幕为二维平面，而非三维，因而需要将三维对象表示为 2D 图像。除此之外，3D 对象通常还包含某些可见边并遮挡其他边。对此，需要进一步考察 3D 空间的表达方式。

17.2　3D 向量

当前，读者可通过一组笛卡儿坐标描述 2D 空间内的一个点，对应坐标沿两个维度（x 和 y）以及与固定原点之间的距离进行测量。与添加一个数字相比，增加一个维度则较为复杂。

17.2.1　添加第三个维度

类似于 2D 几何形状，3D 几何形状可创建一个空间，这涉及定义 3 个正交轴（彼此垂直）和一个原点。其中，对应轴可指向任意方向。根据传统，可将 x 轴表示为“从左向右”方向，y 轴表示为“自下至上”方向。相应地，如果下移两个单位、右移一个单位并上移 3 个单位，该过程对应于向量$(1 –2 3)^T$。

【提示】一类较为常见的 3D 空间定位方式是左手坐标轴。当使用左手坐标轴时，读者的左手握住 z 轴，拇指指向正 z 轴方向，其他手指则按照正 x 和正 y 方向环绕。

三维向量的表达方式与二维向量基本相同，除了增加一个分量之外，向量几何的其他方面均保持不变。实际上，此处依然存在一个附加条件，稍后将对此予以分析。针对 3D 向量的工作方

式，此处考察毕达哥拉斯定理的应用。其中，3D 向量$(x\ y\ z)^T$的大小可记为$\sqrt{x^2+y^2+z^2}$。点积可表示为 3 个分量对之间的乘法运算，如下所示：

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} d \\ e \\ f \end{pmatrix} = ad+be+cf$$

类似于 2D 空间，在 3D 空间内，可通过一点（包含位置向量 **p**）和方向向量 **v** 定义一条直线。根据 t 值，该直线上各点的位置均可表示为 $\mathbf{p}+t\mathbf{v}$。相应地，还可在 3D 空间内定义一个平面。如图 17.1 所示，可使用点 P 以及两个非共线向量 **v** 和 **w**，针对 s 和 t 值，平面上各点可表示为 $\mathbf{p} + t\mathbf{v} + s\mathbf{w}$。另一种方法是选取一点 P 和法线向量 **n**，且向量 **n** 垂直于平面内的全部向量。尽管第二种方法较为高效，但对于某些计算而言，该方法缺少应有的便捷性。因此，本章以及后续章节多采用第一种定义方法。

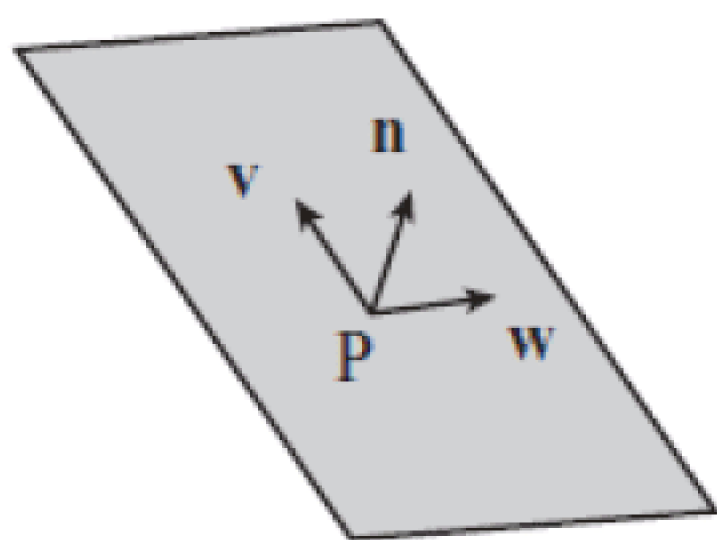


图 17.1　定义一个平面

需要注意的是，作为一个较为有效的方程，法线将平面上的各点关联起来。例如，若法线表示为$(a\ b\ c)^T$，则平面上各点均遵循方程 $ax+by+cz = d$。其中，d 表示为平面与原点之间的垂直距离。该方程可视为直线方程 $ay + bx = c$ 的 3D 等价方程。除此之外，若 **n** 为单位向量，**p** 位于平面上，则有 $d = \mathbf{p} \cdot \mathbf{n}$。

17.2.2　向量（叉）积

3D 点积运算类似于 2D，在三维环境中，存在一种新方法可整合两个向量，即向量积，更为常见的称谓则是叉积。例如，向量 **x** 和 **w** 之间的叉积运算可表示为 $\mathbf{x}\times\mathbf{w}$。与点积运算返回变量值不同，叉积返回一个向量。换而言之，若给定两个向量，则叉积运算所返回的第 3 个向量将与其垂直，最终结果为 normalVector()函数（第 5 章曾有所介绍）的三维等价函数。

与点积相比，向量积的计算过程稍显复杂，如下所示：

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \times \begin{pmatrix} d \\ e \\ f \end{pmatrix} = \begin{pmatrix} bf-cd \\ cd-fa \\ ae-bd \end{pmatrix}$$

为了便于记忆，一种方法是将其视为 3×3 矩阵的行列式：

$$\begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a & b & c \\ d & e & f \end{pmatrix}$$

在当前矩阵中，由于 **i**，**j**，**k** 为基向量$(1\ 0\ 0)^T$，$(0\ 1\ 0)^T$ 和$(0\ 0\ 1)^T$，因而行列式的各元素对应于叉积向量的一个分量。据此，crossProduct()函数如下所示：

```
function crossProduct(v1, v2)
  set x to v1[2]*v2[3]-v2[2]*v1[3]
  set y to v1[3]*v2[1]-v1[1]*v2[3]
  set z to v1[1]*v2[2]-v1[2]*v2[1]
  return vector(x,y,z)
end function
```

叉积运算包含下列明显特征：

- 叉积运算不符合交换律，即 $\mathbf{v}\times\mathbf{w}=-\mathbf{w}\times\mathbf{v}$。另外，叉积运算一般也不符合结合律，即 $\mathbf{u}\times(\mathbf{v}\times\mathbf{w}) \neq (\mathbf{v}\times\mathbf{w})\times\mathbf{w}$。
- 叉积运算符合加法分配律，即 $\mathbf{u}\times(\mathbf{v}+\mathbf{w}) = \mathbf{u}\times\mathbf{v} + \mathbf{u}\times\mathbf{w}$。
- 向量与其自身的叉积结果为 0 向量。
- 对于标量积，若两个向量彼此垂直，则计算结果为 0。
- 若 **v** 和 **w** 为彼此垂直的单位向量，则叉积结果也为单位向量。
- 通常情况下，若 **v** 和 **w** 之间的角度为 θ，则有$|\mathbf{v}\times\mathbf{w}| = |\mathbf{v}|\ |\mathbf{w}|\sin\theta$。
- 两个向量叉积大小可表示为平行四边形的面积，且对应边由向量加以定义。也就是说，该形状的面积 ABCD=$|\mathbf{v}\times\mathbf{w}|$。其中，$\overrightarrow{\mathbf{AB}} = \overrightarrow{\mathbf{DC}} = \mathbf{v}$，$\overrightarrow{\mathbf{BC}} = \overrightarrow{\mathbf{AD}} = \mathbf{w}$。

叉积的方向与轴向的左、右手规则保持一致，即若旋转空间以使输入向量接近于 x 轴或 y 轴，则输出向量也将与正 z 轴对齐。只要基向量保持同一左、右手规则，则叉积结果与其无关。

17.2.3 使用叉积结果

类似于点积运算，叉积运算也可对计算环境进行适当调整。此处，调整工作涉及冗余元素的消除。例如，假设已知某一平面及其法线 **n**，且当前计算需要通过两个面内向量对其进行描述。对此，可采用叉积运算完成这一任务。首先，可选取与 **n** 非共线的任意向量 **w**，并于随后执行叉积运算 $\mathbf{w}\times\mathbf{n}$，这将生成新向量并与 **n** 垂直（同时垂直于 **w**），且该向量位于当前目标平面中。最后，再次执行叉积运算以得到 $\mathbf{u} = \mathbf{v}\times\mathbf{n}$。该叉积运算将得到第二个向量，且同时垂直于 **n** 和 **w**。

【提示】当执行 3D 计算时，有时往往难以区分位置向量和方向向量。这里，位置向量是指原点与既定点之间的向量。通常情况下，方向向量表示为点之间的向量，且各点均由位置向量加以定义。

叉积运算的另一个常见应用是计算旋转轴，稍后将对此加以深入讨论。如图 17.2 所示，位于 **p** 处的箭头指向向量 **v**，且当前任务则是使其指向向量 **w**。通过计算叉积 $\mathbf{v}\times\mathbf{w}$，则可得到垂直于二者的向量，即旋转轴。据此，可围绕该轴向量旋转箭头，以使其指向正确的方向。除此之外，通过计算点积 $\mathbf{v}\cdot\mathbf{w}$，还可进一步得到二者间的旋转角。

点积可视为一类重要计算，该计算往往涉及直线与平面之间的交点。此处，假设直线由点 P 和向量 **u** 予以定义，且平面通过点 Q 和法线 **n** 定义，当前任务是求解 t 值，以使 $\mathbf{p}+t\mathbf{u}$ 位于平面上，

如图 17.3 所示。

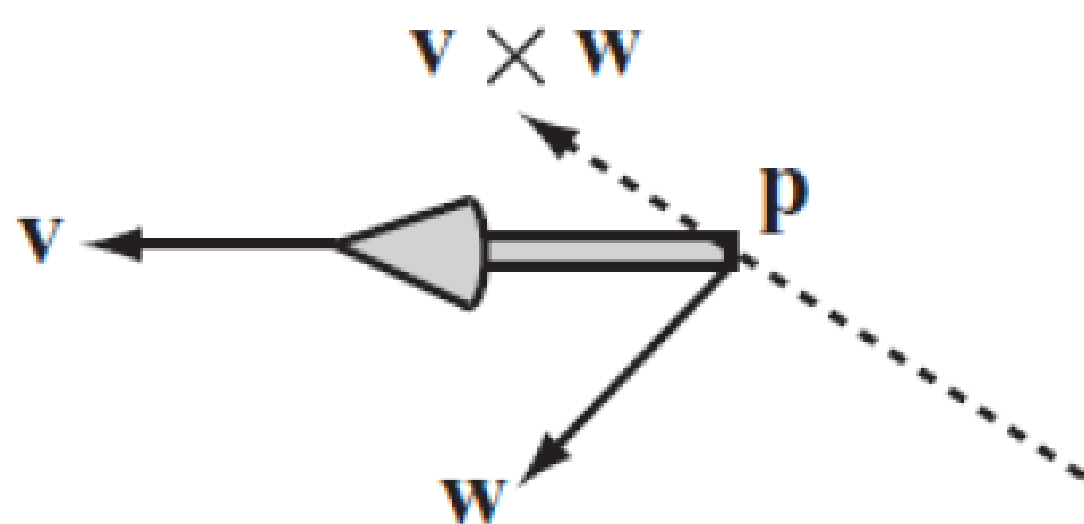


图 17.2　使用向量和标量积计算旋转

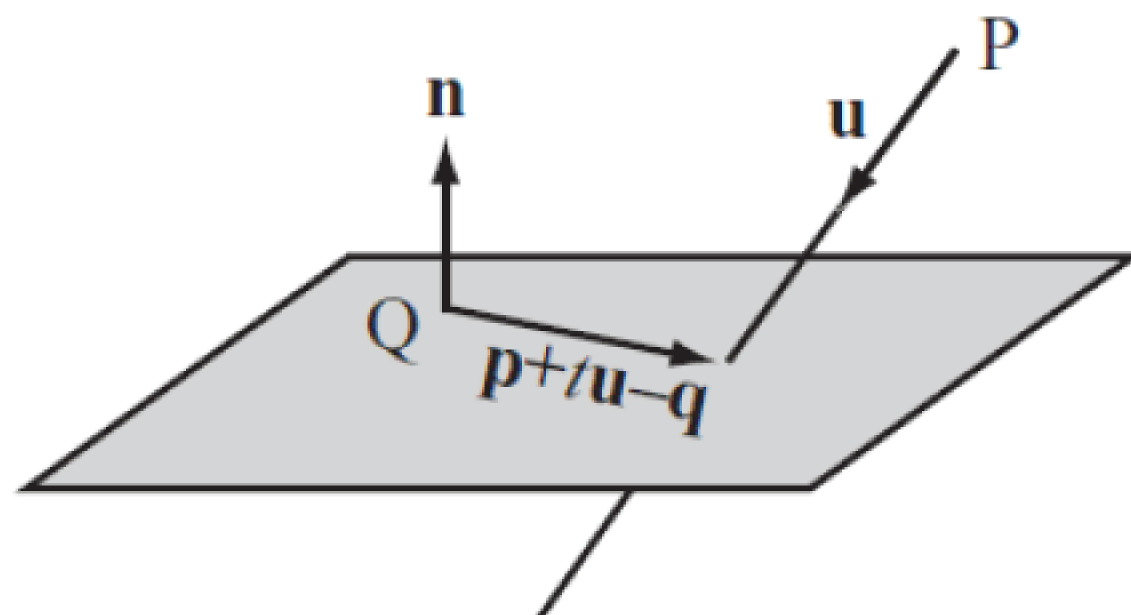


图 17.3　计算直线和平面的交点

相应地，存在多种方法可计算 t 值，一种简单的方法可描述为，针对平面上的任意一点 **a**，**a** − **q** 垂直于 **n**，且有：

$$(\mathbf{p} + t\mathbf{u} - \mathbf{q}) \cdot \mathbf{n}=0$$

由于标量积遵循结合律，因而有$(\mathbf{p}-\mathbf{q}) \cdot \mathbf{n}=t\mathbf{u} \cdot \mathbf{n}$。最终则可获得如下结果：

$$t = \frac{(\mathbf{p}-\mathbf{q})\cdot\mathbf{n}}{\mathbf{u}\cdot\mathbf{n}}$$

正如期望的那样，由于直线与平面之间不存在交点，因而当 **u** 和 **n** 彼此垂直时，该方法无效。否则，这一方法则快捷、有效。

同时，读者还可采用类似的技术计算两个平面之间的交线。相应地，若两个平面分别通过点 P 和 Q 以及法线 **n** 和 **m** 加以描述，则当前任务为计算位于两平面上的直线，如图 17.4 所示。

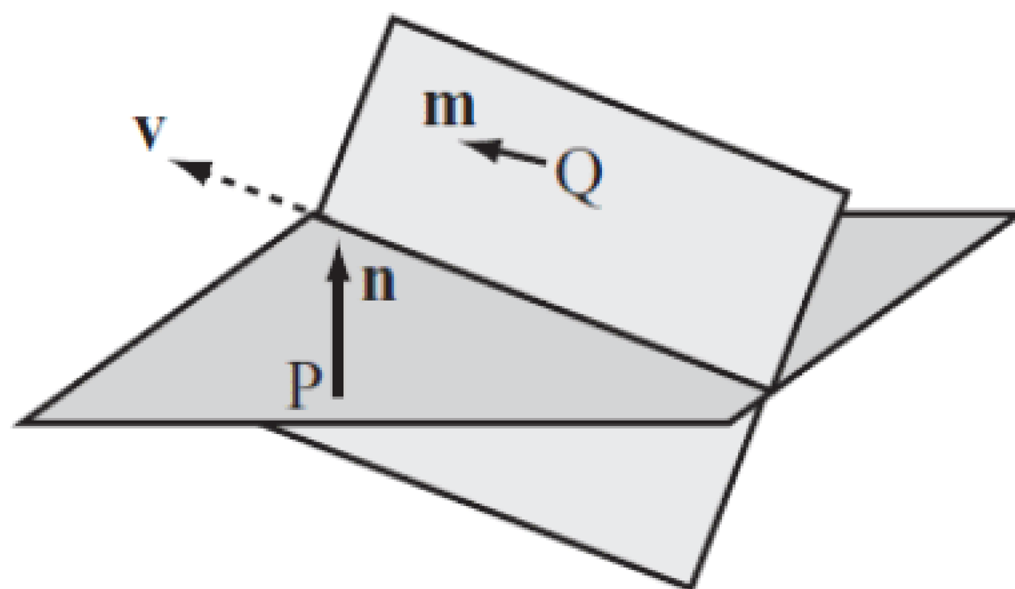


图 17.4　两个平面的交线

由于直线位于两个平面上，因而皆垂直于 **n** 和 **m**。鉴于该垂直性，可将 **n**×**m** 作为其方向向量 **v**。至此，仅需计算直线上的单一点。当计算该点时，可使用前述计算结果。相应地，可在第一个平面内选取任意向量（较好的选择是 **n**×**v**），并查看直线与第二个平面之间的交点（该直线沿当前向量穿越 **p**）。linePlaneIntersection()和 planePlaneIntersection()函数封装了上述操作，且第一个函数体现了直线与平面之间的交点计算，如下所示：

```
function linePlaneIntersection(linePt, lineVect, planePt, planeNormal)
  set d to dotProduct(lineVect,planeNormal)
  if d=0 then return "no intersection"
  set v to linePt-planePt
  return dotProduct(v,planeNormal)/d
end function
```

第二个函数则定义了平面间的相交计算，如下所示：

```
function planePlaneIntersection (pt1, normal1, pt2, normal2)
  set v to crossProduct(normal1,normal2)
  set u to crossProduct(normal1, v)
  set p to linePlaneIntersection(pt1, u, pt2, normal2)
  if p="no intersection" then return p
  return array(p,v)
end function
```

17.2.4 齐次坐标

虽然仅需通过 3 个坐标表达三维空间，但在实际操作过程中，往往还需要使用到第 4 个坐标，具体原因将在第 18 章中介绍。当前，可在 x，y，z 坐标中添加第 4 个坐标 w，对应结果为四维向量，该向量遵循前述二维和三维向量的运算规则。

很多时候，第 4 个坐标十分有用，尽管其设置稍显随意且缺乏明晰的含义。这里，w 坐标称作齐次坐标，术语“齐次”具有“包含某一类似维度”的含义。对于位置向量，w 设置为 1；对于方向向量，w 则设置为 0。为了进一步阐述齐次坐标的工作方式，考察如下方程：

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} - \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix} = \begin{pmatrix} x-a \\ y-b \\ z-c \\ 0 \end{pmatrix}$$

位置向量之间形成的向量，其 w 分量为 0——当前无须刻意强调这一点，在后续讨论中将会发现，基于齐次坐标的向量加法并不等同于包含正规坐标的向量加法运算。

同样，二维环境有助于理解三维计算中的齐次坐标。如图 17.5 所示，二维平面通过 x'和 y'加以定义，该平面置于包含轴 x，y 和 w 的三维空间中，并与 $w = 1$ 平面保持一致。最终，该平面内的任一点(x', y')与 3D 空间内的$(x, y,1)$点完全一致。

然而，并非仅是平面上的点数据可映射至 2D 空间中，通过投影处理，全部 3D 空间均可“压缩”至某一平面上。如图 17.6 所示，针对任意点 P，可在 3D 原点与 P 之间绘制一条直线，并计算与平面之间的交点。

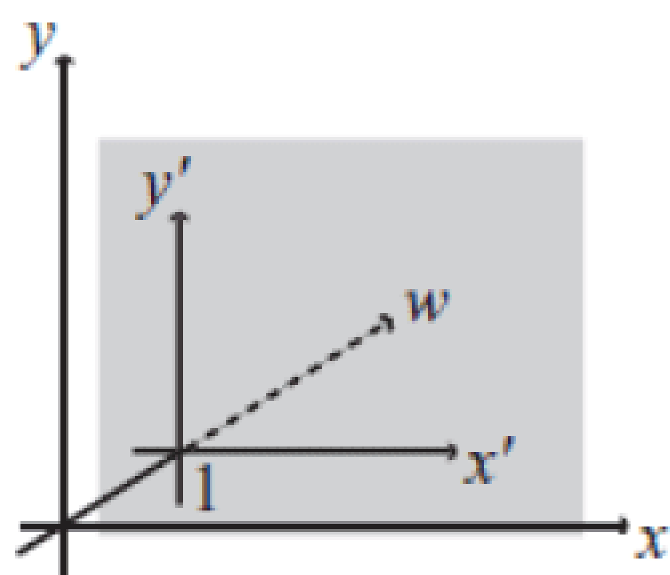


图 17.5　二维环境下的齐次坐标

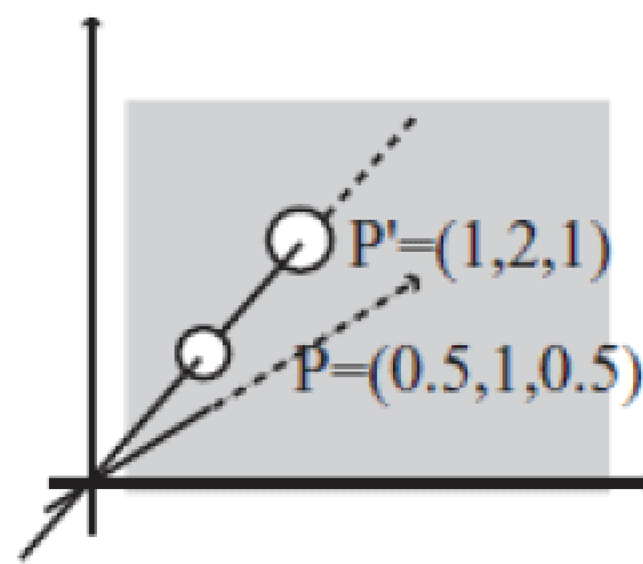


图 17.6　向平面投影 3D 齐次空间

除此之外，读者甚至还可对包含 $w = 0$ 的齐次点执行上述操作。虽然直线 OP 平行于当前平面，但依然可认为该直线与平面相交于无穷远处，也就是说，直线沿平行于 OP、位于平面内的直线与当前平面相交于无穷远处。除此之外，对应于(x, y, w)的点可表示为$\left(\frac{x}{w}, \frac{y}{w}\right)$。从严格意义

上讲，齐次点(x, y, w)与$\left(\frac{x}{w},\frac{y}{w},1\right)$之间并无明显差别。当从数学角度上看待这一问题时，二者间视为彼此等同。最终结论可描述为，齐次坐标具有尺度不变性（scale-invariant）。若通过某一常数因子对其执行乘法运算，二者将得到同一计算结果。

随着齐次坐标的引入，可再次审视 2D 环境中两条直线之间的交点问题。对此，基于向量$\begin{pmatrix}p\\q\end{pmatrix}$且穿越$(a, b)$的直线可表示为如下等式：

$$a + tp = x$$
$$b + tq = y$$

进而有：

$$\frac{x-a}{p}=\frac{y-b}{q}$$
$$qx - py - aq + bp = 0$$

通常情况下，2D 环境中的直线可表示为 $ax + by + c = 0$ 这一形式。当采用齐次坐标时，该式可定义为 $ax + by + cw = 0$。通过对两种形式的比较可知，2D 环境中的直线对应于齐次坐标空间中的平面。若通过该方式表达两条直线，则可根据下列方案计算其交点：

$$ax+by+cw=0$$
$$px+qy+rw=0$$
$$\frac{x}{w}=\frac{br-cq}{aq-bp}$$
$$\frac{y}{w}=\frac{cp-ar}{aq-bp}$$

根据上述操作，即可得到齐次坐标$(br - cq, cp - ar, aq - bp)$。需要注意的是，若 $aq - bp = 0$，则两条直线处于平行状态。通过观察可知，标准坐标形式中，对应方程此时无解；而在齐次坐标中，两条直线相交于无穷远处的一点（$w = 0$）。进一步讲，该点的位置向量可通过下列叉积予以确定：

$$\begin{pmatrix}a\\b\\c\end{pmatrix}\times\begin{pmatrix}p\\q\\r\end{pmatrix}$$

叉积的意义体现于可生成 3D 空间内的平面法线，进而得到平面相交结果。这里，该结果对应于 2D 直线的交点。

在后续章节中，齐次坐标还将再次被提及，例如 3D 环境中的应用方式。17.3 节将讨论 3D 渲染中的投影面，且齐次坐标与投影面之间关联紧密。

17.3　渲 染 机 制

3D 场景与 2D 图像之间的转换过程称作渲染，后续章节将依次对此予以介绍。当前仅讨论

某些预备知识，且与渲染过程所涉及的物理介质相关。

17.3.1 投影平面

关于投影面，首先须考察眼睛与周围物体反射光线之间的接收方式。由于光线通常以直线方式传输，若从眼中向外投射一条直线，则眼中看到的是这条直线上的第一个物体，且视野包含了多条直线所形成的场景，对应直线以某个角度从眼中投射。

当根据 3D 场景予以考察时，假设观察者处于静止且直视前方。此时，观察者的可见区域（即前方侧向、上方、下方区域）构成了视域，如图 17.7 所示。除此之外，图 17.7 还显示了近平面和远平面。

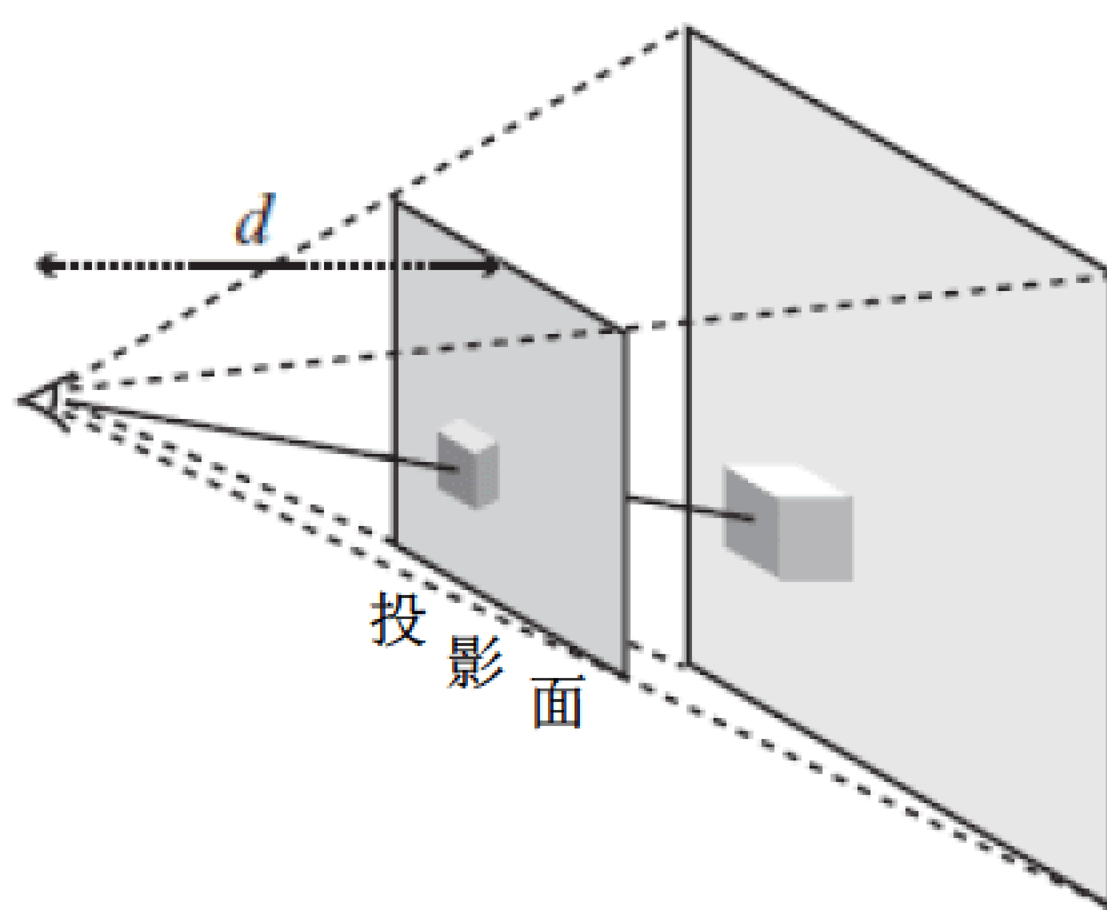


图 17.7 视域和视锥体

【提示】当描述 3D 几何形状时，难点之一则是在 2D 页面上绘制清晰的示意图，图 17.7 显示了位于投影面上的经转换后的 3D 立方体对象。

视锥体中的平面可视为 3D 与 2D 图像转换时的主要“设备”。在图 17.7 中，假设观察者在距离 d 处的投影平面上观察当前场景世界，当计算平面各点上的观察结果时，可从观察者处绘制一条直线，并查看碰撞结果——该过程称作光线跟踪。对于实时动画，可计算各对象于空间内的位置、向观察者绘制一条直线，并获取该直线与各平面的相交位置。基于光照和纹理的准确模拟过程相对复杂，但主要过程大抵如此。

如图 17.8 所示，当计算与空间某一特定点对应的、位于投影平面上的一点时，假设观察者朝向方向 **n**，且目标点位于点 **p** 处。类似地，假设投影平面与观察者（位于点 O 处）之间的垂直距离为 d。

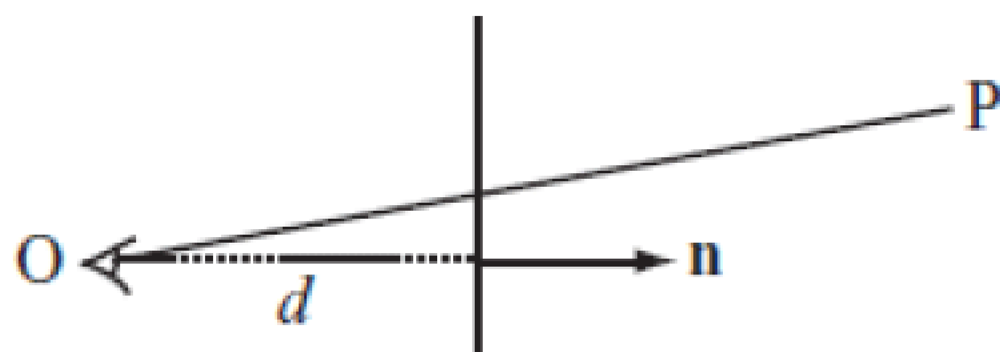


图 17.8 将一点投影至投影面上

图 17.8 显示了横截面示意图，并在点 P 和观察者之间绘制一条直线，该直线在某一未知点

处穿越投影平面。该情形基本等同于图 17.7 所述场景。

从数学角度上讲，图 17.8 中的平面法线等同于观察者的朝向。其中，平面（该平面绘制于绘制于屏幕中心）参考点可记为 $\mathbf{o}+d\mathbf{n}$，并假设 $\mathbf{n}$ 为标准化向量。因此，与 P 对应的投影平面点可通过直线（始于点 P 且方向为 $\mathbf{o}-\mathbf{p}$）与该平面之间的交点加以确定，即点 $\mathbf{p}+t(\mathbf{o}-\mathbf{p})$。其中：

$$t=\frac{(\mathbf{p}-(\mathbf{o}+d\mathbf{n}))\cdot\mathbf{n}}{(\mathbf{o}-\mathbf{p})\cdot\mathbf{n}}=-d\frac{\mathbf{n}\cdot\mathbf{n}}{(\mathbf{o}-\mathbf{p})\cdot\mathbf{n}}-1=\frac{d}{(\mathbf{p}-\mathbf{o})\cdot\mathbf{n}}-1$$

需要注意的是，由于 $\mathbf{n}$ 为标准化向量，因而有 $\mathbf{n}\cdot\mathbf{n}=1$。

待投影平面计算完毕后，则可在计算机屏幕上以正确的尺寸绘制相关内容，即视口。其中，存在多种方法可实现这一任务。一种方案是确定比例规格，例如，投影面的一个单位对应于屏幕上的一个像素。如图 17.9 所示，一类较为常见的视口定义方式是，确定观察者视域的最大角度，例如 θ 角。在某些场合下，全部视角定义为 2θ。

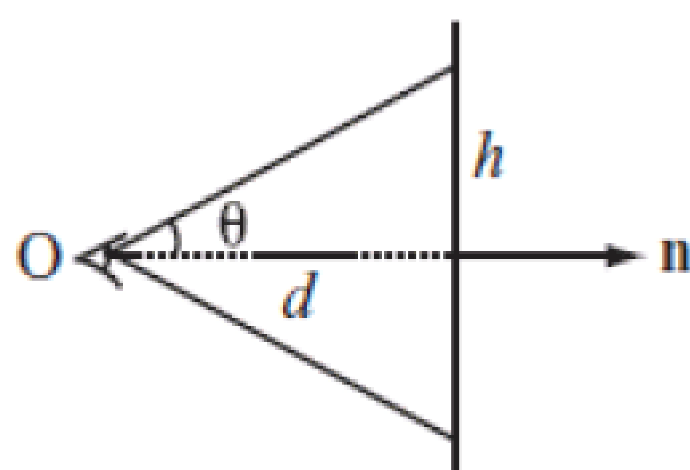


图 17.9　确定视域

由于 $\tan\theta=\frac{h}{d}$ 以及投影面最高点的高度值均为已知项，因而可形成一个比例值以绘制当前屏幕。

然而，由于无须使用到 d 值，该方案可能过于复杂。实际上，全部所需项为屏幕上的图像尺寸以及视角。为了获取正确的计算结果，可在适当处设置投影平面，并于随后使用相似三角形计算。

相应地，可计算沿相机向量方向上的、点与观察者之间的距离，即 $(\mathbf{p}-\mathbf{o})\cdot\mathbf{n}$，以及垂直位移 $(\mathbf{p}-\mathbf{o})\cdot\mathbf{u}$。其中，$\mathbf{u}$ 表示为相机的“向上”向量。若投影平面的高度值为 h 且有 $d=\frac{h}{\tan\theta}$，则该点在屏幕上的投影高度为 $\frac{h_{\max}\cdot(\mathbf{p}-\mathbf{o})\cdot\mathbf{u}}{\tan\theta(\mathbf{p}-\mathbf{o})\cdot\mathbf{n}}$。函数 pos3DToScreenPos()涵盖了前述各项操作，并将空间中的一点转换至屏幕上。该函数的参数包括观察者的位置和法线、视角、屏幕高度以及观察者的“向上”向量，如下所示：

```
function pos3DToScreenPos(pt, observerPos, observerVect,observerUp, fov, h)
  set observerRight to crossProduct(observerUp, observerVect)
  set v to pt-observerPos
  set z to dotProduct(v,observerVect)
  set d to h * tan(fov)
  set x to d*dotProduct(v,observerRight)/z
  set y to d*dotProduct(v,observerUp)/z
  return vector(x,-y)
end function
```

pos3DToScreenPos()函数返回相对于 3D 视口中心位置的屏幕上的点位置，其中，y 坐标沿下方予以测算。在实际操作过程中，鉴于此类数据往往通过预计算方式得到，因而函数的效率可获得大幅提升。需要说明的是，对于观察者后方的数据点，其计算过程稍显复杂。实际上，读者可将这一部分工作交与 3D 图形卡，稍后将对此加以讨论。

如前所述，当讨论图像的显示方式时，相对于观察者的可见点集称作视见体。在图 17.7 中，金字塔截去顶端后形成了部分视见体，该形状包含 6 个面，各条边通过视域并以某一角度予以确定。视见体的前、后面可随意设定，尽管从理论上讲观察者可看到无穷远和无穷近处，但在渲染过程中，依然需要在特定距离剪裁空间，并采用雾效果模糊化消失于视野的远距离物体。

在屏幕投影方程中，尽管改变 **o** 和 θ 可影响屏幕上的图像，但该过程包含了多种实现方式。观察者处于运动状态且 θ 保持不变将移动当前视锥体，并线性地改变图像的比例。当调整视域时，视锥体的形状将产生变化，这与改变相机镜头的角度十分类似。为模拟环境选取正确的视域须设置正确的参数，就像导演或电影摄影师须花费很长时间选择心仪的镜头那样。此事无关正确性，仅是不同的视觉效果而已。当观察视角与计算机用户和屏幕之间的视角相同时，对应观感最为自然。当然，这还取决于用户与屏幕之间的距离，以及屏幕的分辨率。

17.3.2　透视

透视技术用于表达平面上的对象，该技术在文艺复兴时期曾通过数学方式予以定义，但透视基本原理的基本应用则先于文艺复兴时期。当采用数学定义后，透视可用于纸面上的 3D 场景绘制。尽管该技术对于绘画而言十分有效，但在屏幕上渲染图像时，仍需采用某种转换技术。

当绘制透视场景时，可于开始阶段绘制一条水平线，且位于观察者的眼睛高度。水平线上的各点表示为消失点，进而体现了沿某一直线（基于观察者的垂直轴）的无穷远处的对象。图 17.10 显示了一类经典示例，即延伸于远方的公路。

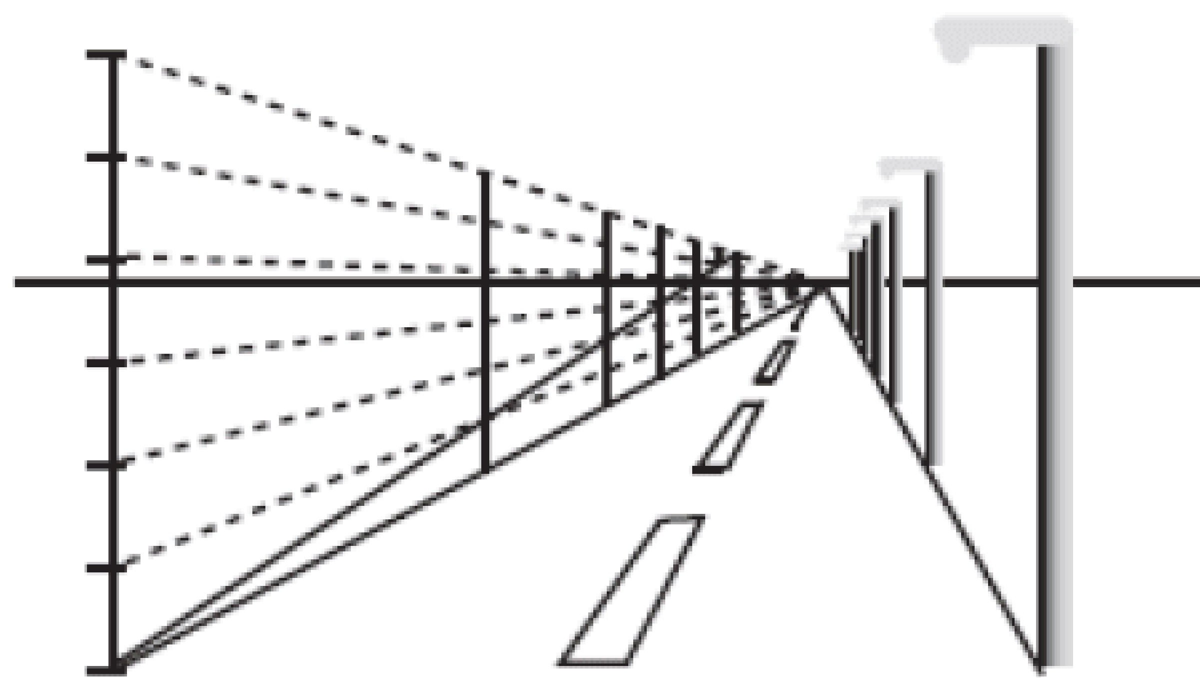


图 17.10　透视绘图法

除了公路之外，图 17.10 还绘制了大量的灯柱。若读者沿公路行进，则间隔排列的灯柱类似于多条直线。其中，左侧的虚线显示了基本绘制技术。另外，图中显示了两种类型的直线，始于第一个灯柱并交汇于水平线一点处的直线称作正交直线，而用于表示灯柱的直线则称作横向线。

当描述透视效果时，可将第一个灯柱的横向线划分为多个均等部分，并通过正交线将各部分连接至消失点。随后，可从第一个灯柱的下方与远景灯柱的上方之间绘制一条直线，其余灯柱的位置则与该直线与正交直线的交点保持一致。

从数学角度上讲，对象图像尺寸以距离指数方式缩减，如下所示：

$$图像高度 = 高度 \times e_0^{-k(d-d_0)}$$

其中，k 表示常数，d_0 表示对象距离，其中，图像高度与实际高度相等。公式中的 e 值并无特殊之处，实际上，若适当调整 k 值，任何基数均可实现相同的结果。

在计算机 3D 引擎中，投影方案并非十分重要，其原因在于，若观察者处于移动状态，则多项数据需要重新计算。然而，当与简单场景以及固定观察者协同工作时，该基数则易于使用并可快速地生成 3D 效果。绘制和渲染之间的差别涉及透视绘图中的 k 与 d_0，以及标准 3D 渲染中 d 和 θ 之间的关系。

17.3.3　正交投影

虽然标准的透视投影可视为 3D 场景中的常见方法，但在某些场合下，其他方法则更为有效，特别是建筑制图和工程制图。

标准的透视投影有时也称作中心投影。当采用中心投影时，若远离投影平面，则会出现较为奇特的现象。如图 17.11 所示，当 d 增加且 θ 减小时，屏幕上的对象通常呈现为相同尺寸，而视见体则演变为一个简单的盒体。

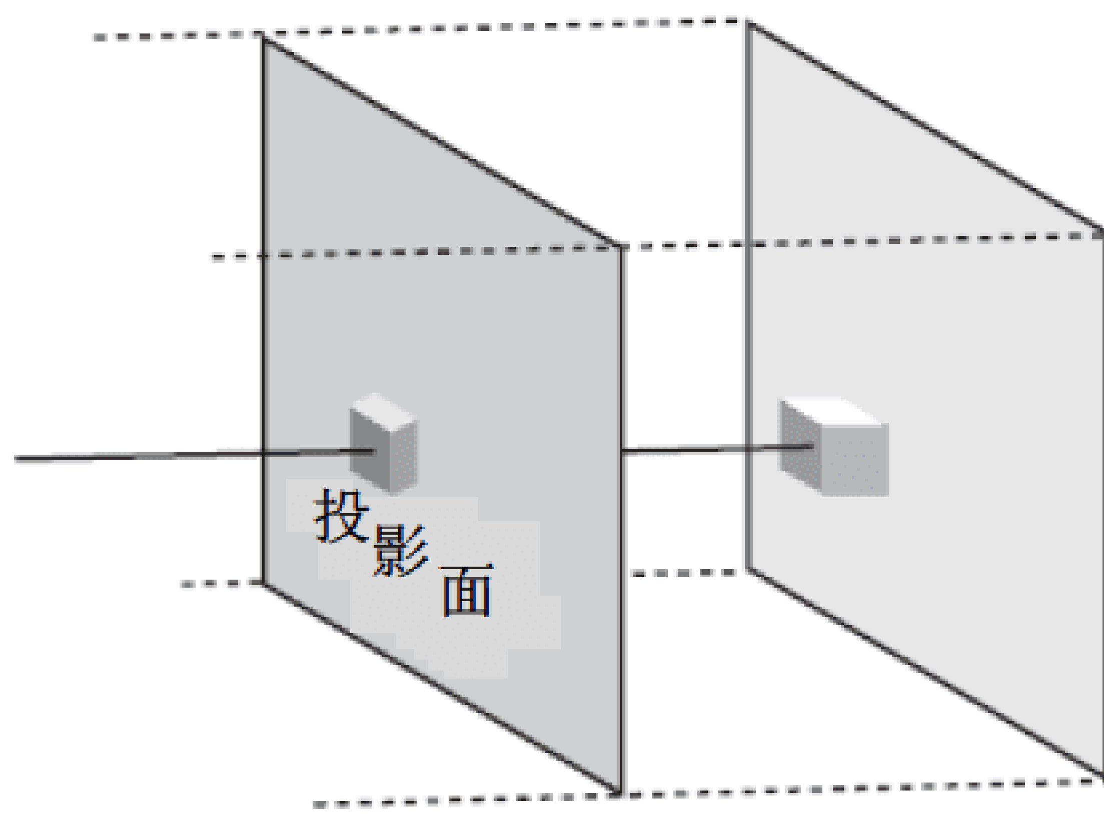


图 17.11　正交投影

如图 17.11 所示的图像称作正交投影。若考察对象处于不同的角度，该过程也称作斜视图、轴测投影或者等轴测投影，但均等同于正交视图。在正交视图中，对象与观察者或其他对象之间的距离不会对其尺寸产生任何影响。然而，这将影响到对象在屏幕上的位置以及绘制顺序。否则，全部工作仅需知晓视口的尺寸。

在游戏制作中，较为常见的正交视图为等轴测视图。当以某一角度（对象各边与投影面呈相同角度）以及正交投影方式俯视对象时，对象均以相同长度予以呈现（本书第 5 部分将简要介绍等轴测游戏）。

另一类投影技术同样值得关注，该技术围绕观察者采用了投影球体或圆柱体，而非投影面。某些时候，该技术可呈现更为真实的观感。毕竟，人类通过可转动的眼球观察这个世界，而非一扇窗户。然而，投影球体可产生某些奇怪的效果，例如地球的墨卡托投影，并可视为一类较为常见的行星贴图。在墨卡托投影中，可将地球置于纸状圆柱体内部，并从中心位置处发射光线。投

射于圆柱体上的阴影构成了一幅贴图，随后可将纸张展开。

该方式可较好地处理球状对象的贴图问题，例如地球的平面贴图。然而，此类方案也包含相关问题，例如，距离测算通常较为困难。另外，距离赤道越远，则须更多地展开贴图，直至北极处无限展开贴图，即圆柱体的高度趋于无穷大。因此，在地图上，格陵兰岛与其实际尺寸并不成比例。若采用此类投影环游世界，则最终路线将会令人迷惑不已。

17.4 光 线 投 射

在 2D 图像操作中曾有所提及，沿一条光线获取对象列表通常十分有效。这里，光线可视为在某一方向上无限延伸的、且包含一个终止点的直线。第 10 章曾讨论了光线与平面之间的交点计算方式，并于随后介绍了光线与简单形状以及多边形网格之间的碰撞计算。当前阶段将考察某些具体应用，并假设读者可通过 3D 引擎计算光线。

光线投射在多种场合下均十分有用，以下两项内容体现得尤为明显。首先是用户交互体验，例如，用户单击 3D 场景进而计算其下方对象。其次是碰撞检测计算，例如，光线投射可用于确定前方障碍物。

17.4.1 通过 3D 引擎计算路径上的对象

光线投射的基本理念是向实时引擎发送查询，并传递起始点和方向向量。随后，查询结果返回与光线相交的模型列表。取决于具体的 3D 引擎，查询操作可包含某些优化方案。例如，某些时候，可定义所返回的最大模型数量、所需检测的模型列表以及光线的最大长度。除此之外，其他详细内容还包括碰撞点、碰撞法线以及所单击的纹理坐标。

地形跟踪系统可视为一类较好的示例，其中，对象沿高度变化的地面行进。这里，假设地面由连续的三角形网格构成，并构建一个 4×4 车辆对象行驶于地面上。这里的问题是，如何使车辆真实地行驶于崎岖不平的路面上？对此，需要了解车轮的高度以及车辆的对应方向。

地面高度的一种计算方法是将地面信息存储为 2D 高度图，该方案提供了快速的信息访问机制，但与此同时，也需要使用到较大的存储空间。另一种方法则是从各车轮处向下投射光线，并根据碰撞点调整车辆方向，取决于 3D 引擎的速度、显卡的速度以及地形几何形状的复杂度，该方案可实现较快的计算速度。除此之外，这一方案还具有其他优势，例如可与随时间变化的地形协同工作，如水波。

类似的示例还包括第一人称射击游戏（FPS），其中，玩家或敌方角色通常以直线方式发射子弹。此时，快速的光线投射碰撞检测可确定相应的击中位置。

需要说明的是，该技术并不适用于下列场合：对象尺寸 3～4 倍于碰撞网格，或者目标碰撞对象较小。对此，需要投射大量的光线以判断路径的清晰度。当然，这里存在一类变通方案，例如，若对象间的尺寸相差较大，则可通过“代理”形状执行近似碰撞处理，例如球体或包围盒。

17.4.2　拾取、拖曳以及投掷操作

交互体验同样需要光线投射操作，例如，游戏玩家单击屏幕以选取 3D 空间内的对象。对应问题可描述为，如何确定所选对象？

通过选择屏幕上的一点，用户即选取了基于某一条光线的角度值。实际上，用户当前正在执行 3D 点-屏幕的逆向操作。确定该点的一类最为简单的方法是使用投影屏幕法，进而计算鼠标位置处的特定 3D 点。随后，可通过该点创建方向向量。如前所述，对此需要使用到视域角度 θ 以及视口的高度值 h。因此，与投影面之间的距离可记为 $d=\dfrac{h}{\tan\theta}$。

待该距离计算完毕后，则可知相机的前向和向上向量，因而可快速计算鼠标位置对应的数据点。screenPosTo3DPos()函数执行了上述计算过程，如下所示：

```
function screenPosTo3DPos(viewportPos, observerPos,observerVect, observerUp,
                              fov, h)
  set observerRight to crossProduct(observerUp, observerVect)
  set d to h / tan(fov)
  return observerVect*d - observerUp*viewportPos[2] + observerRight* viewportPos[1]
end function
```

当前，光线投射的准备工作实现完毕。其中，初始点为相机位置，对应方向为与计算点之间的向量，以此可知所选的具体模型对象。

这里，假设需要将对象拖曳至某一新位置，尽管空间内存在 3 个可移动维度，但鼠标存在仅可体现两个维度。无论如何，用户的移动需要转换至 3D 空间内。

运动的转换方式取决于特定的操作环境，若模型对象为部分平面，情况又当如何？该对象可能是地面上的对象、滑块或墙面上的图像。对此，两个方向已然足够，全部工作仅需限制某一特定方向上的运动。

如果对象在空间内自由移动且不包含旋转行为，则最佳方案是将鼠标移动与某一按键结合使用。在程序实现中，用户在移动鼠标时可按下 Shift 键，这将使得对象在相机的 xz 平面内移动，而非 xy 平面。另外，若拖曳行为仅出现于当前 xy 平面，则用户可围绕对象自由移动，进而改变视平面或提供同一对象的多个视角。最后，若模型对象包含旋转行为，则鼠标移动应对应于两轴方向上的旋转效果。

对此，首先需要对当前平面予以限制，这可视为前述操作的扩展行为。当鼠标移动时，可从其当前位置至目标平面处投射一条光线，并可作为拖曳对象的最新位置，并适当偏移其高度值或半径值。需要说明的是，对应位置无须位于某一平面内，并可能是粗糙地形或其他对象。在撞球游戏中，这将涉及虚拟球体的放置，并以此标明可能出现的碰撞位置。

针对更为高级的变化版本，还可“模拟”光线位置，以使用户认为正在拖曳其中心位置，而非底部，图 17.12 显示了这一工作方式。首先，当对象位于屏幕坐标系时需要计算其底部相对于中心位置的偏移量；随后，可将该偏移量应用于当前鼠标位置上，进而获得地面位置——当前，用户可拖曳对象并使其位于该位置上。期间，用户并未精准地拖曳对象的中心位置，但用户并不

会对此有丝毫的察觉。

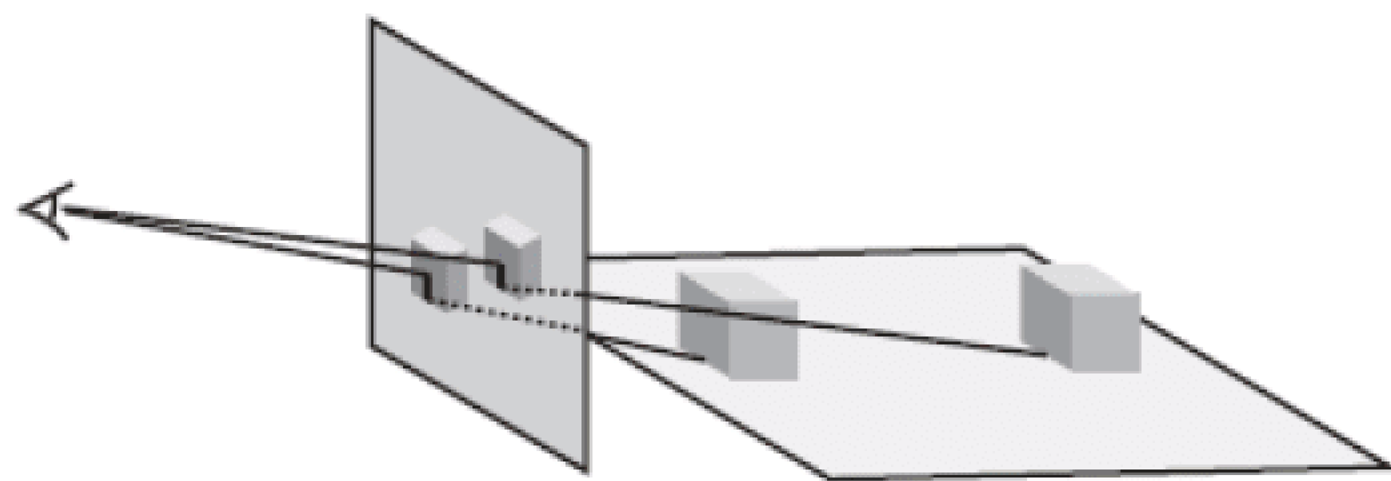


图 17.12　沿平面拖曳对象

在空间内自由地拖曳对象可视为上述操作的扩展行为，且通常需要约束于某一平面。关于平面，此处则存在相对宽泛的选择余地。通常情况下，尽管最为自然的方法是使用基于相机的平面法线，但读者也可尝试使用“向上”向量的法线，或其他更为便捷的向量。

当用户操控鼠标以使对象旋转时，相关问题也会随之产生。某些简单的模拟环境使得玩家可自身旋转，例如鼠标驱动的 FPS 游戏。其中，鼠标移动对应于相机的旋转。

该问题易于解决，且鼠标移动与期望运动之间存在直接的映射关系。在各时间步内，用户可旋转相机以使其指向光线方向（第 18 章将讨论如何旋转对象，以使其指向特定方向）。对此，需要确定仅当鼠标移动时场景是否处于旋转状态。期间，用户需要在鼠标移动时将指针位置重置于屏幕中心位置（因此，当采用该方案时，应隐藏鼠标指针）。

当通过鼠标并以相机视角旋转某一对象时，情况则变得稍加复杂，且操作间不存在明确的转换行为。在开始阶段，假设旋转对象时，被单击的对象点处于鼠标控制下。对此，仅需知晓光线的交点即可。此处，其难点在于“向上”方向的处理方式。第 18 章将讨论到，可针对某一操作点制定“向上”向量。

针对上述观察，下面考察其具体应用方法。这里，旋转对象包含位置向量 $\mathbf{p}$，且单击时对应点的位置向量表示为 $\mathbf{q}$。当前，鼠标移动后相对于观察者定义一条光线，即 $\mathbf{o} + t\mathbf{v}$。这里，需要使当前对象指向光线方向。

为了获取正确的指向，需要计算光线与球体之间的交点，该球体位于 $\mathbf{p}$ 处，且半径为$|\mathbf{q}-\mathbf{p}|$。若存在对应交点，即可得到对象所指向的最新位置；否则，鼠标则位于对象的外部区域。此时，可持续执行旋转操作，抑或使其尽可能地接近于鼠标的朝向。另一种设置方法则是使其朝向垂直于向量 $\mathbf{v}$ 的方向，并穿越由 $\mathbf{v}$ 和 $\mathbf{p}-\mathbf{o}$ 构成的平面。

此时，“向上”向量应如何处理？一种选择方案是使用相机的“向上”向量，然而，当旋转对象以使其与该向量对齐时，这一方案将会导致问题的出现。另一种方案则是使用旋转轴，但这将导致对象缺乏稳定的旋转状态。对此，一类较好的折中方案则是在相机方向向量和点的移动向量之间采用叉积计算。

17.5　本 章 练 习

【练习 17.1】试编写函数可绘制三维立方体，并可执行背面剔除操作。尽管可通过平面法线确定立方体的可见表面，但具体内容稍后将做深入讨论。当前，可将重点放在投影方面上。对此，

读者可通过叉积运算确定立方体的某一表面是否可见。另外，读者还可使用该函数围绕其中心位置旋转，并考察不同投影技术以及视域所产生的视觉效果。

17.6　本 章 小 结

本章引入了三维空间及其工作方式，并可通过二维对象中的相关概念理解第三个维度。当然，由于引入了额外的自由度以及观察者（尽可看到部分场景），因而计算过程也将变得更加复杂。

第 18 章将进一步考察 3D 空间的工作方式，并对第 5 章所讨论的矩阵数学进行适当的扩展，进而处理更为复杂的运动和转换操作。

至此，读者应掌握如下内容：

- 如何将向量扩展至三维空间内。
- 叉（向量）积的含义和应用。
- 如何计算直线和平面之间的交点和交线。
- 术语“齐次坐标”的含义及其相交计算的应用方式。
- 如何使用不同的投影创建 3D 引擎。
- 如何使用光线投射在 3D 空间内构建用户体验。

第 18 章　转 换 操 作

本章包含如下内容：

- 概述。
- 描述空间位置。
- 转换操作的应用。

18.1　概　　述

读者已经了解了 2D 矩阵的应用方式，进而描述 2D 空间内的转换序列。本章将考察 3D 转换的使用方法，并根据各自关系创建对象。

18.2　描述空间位置

3D 场景世界由称为“模型”的对象构成。其中，模型可视为以三角形方式连接在一起（进而形成网格）的点集构成。为了计算模型的具体位置，3D 引擎将其视为称为“节点”的独立对象。同样，相同的方法也适用于描述光源、相机以及其他 3D 空间成员的位置。

18.2.1　位置、旋转和缩放

当描述一个节点时，应获取其位置、旋转以及缩放等信息。其中，旋转和缩放可通过转换矩阵予以处理，而位置则与向量类似。针对转换组合的可视化操作，最为简单的方式是作用于原始模型上的单一处理过程，分别包含缩放操作、围绕原点的旋转操作以及至当前点的平移操作。

上述转换统称为仿射转换，也就是说，两条直线在转换前处于平行状态，则转换后二者仍处于平行状态。同样，剪切和反射操作亦视为仿射转换。实际上，任何可通过矩阵执行的转换行为均为仿射操作。旋转、平移和反射操作则相对严格，并可视为刚体转换，由于同时隶属于仿射转换，因而空间直线间的角度保持不变。换而言之，尽管顶点之间的相对位置发生变化，但对应角度值保持固定，但这对缩放操作并不成立，如图 18.1 所示。

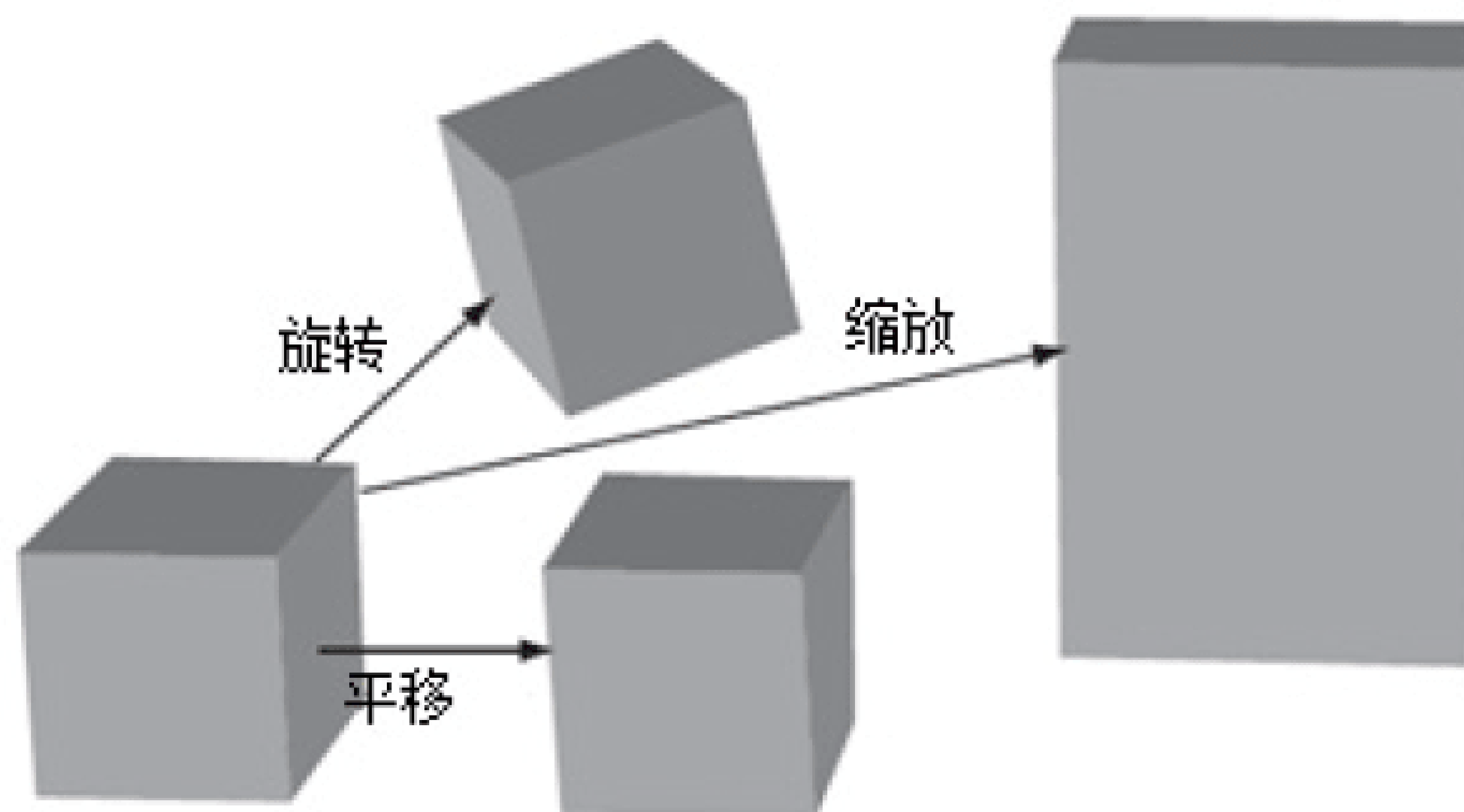


图 18.1　3 种基本的转换操作

缩放操作的可视化过程则相对简单，基于常数因子的缩放操作须将各个位置向量乘以缩放因子。从矩阵数学角度来看，当前矩阵将乘以矩阵 $n\mathbf{I}$。其中，$\mathbf{I}$ 表示单位矩阵。通常情况下，各个方向可按照不同幅度进行缩放，因而向量需要乘以下列矩阵：

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$$

任何缩放转换均可通过该方式进行计算。例如，方向(1　1　0)上的缩放因子 $\sqrt{2}$ 等价于下列矩阵：

$$\begin{pmatrix} \sqrt{2} & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

与缩放操作相比，旋转则更为复杂。在二维环境中，围绕原点的旋转行为相对直观，并可通过某一角度值加以描述。当引入第三个维度后，通常需要确定一个旋转轴。空间内的任一向量均可作为旋转轴，在前述撞球游戏中，考虑到上旋和侧旋问题，任一旋转均可分解为围绕基向量的、更为简单的旋转行为。

对于围绕某一基向量的旋转行为，对应矩阵如下所示：

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

结合上述 3 个矩阵可生成一个通用矩阵，如下所示：

$$\mathbf{R} \equiv \mathbf{R}_z\,\mathbf{R}_y\,\mathbf{R}_x = \begin{pmatrix} \cos\xi & \sin\xi & 0 \\ -\sin\xi & \cos\xi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

【提示】这里，符号“≡”表示“定义为”或“等价于”，即恒等号。

矩阵乘法运算貌似复杂，但实际过程并非如此艰难。假设通过 $\mathbf{R}$ 对向量 $\mathbf{i}$，$\mathbf{j}$，$\mathbf{k}$ 进行转换，结果矩阵中的列向量分别为 $\mathbf{u}$，$\mathbf{v}$，$\mathbf{w}$。旋转操作的特征之一即是点及其图像之间的向量应与旋转

轴垂直。若执行叉积计算$(\mathbf{u}-\mathbf{i})\times(\mathbf{v}-\mathbf{j})$，则新向量 **a** 应与二者垂直，且定义为旋转轴。

除此之外，还可通过垂直于 **a**、**i** 和 **v** 的分量进一步计算旋转角 α。也就是说，将 **i** 和 **v** 投影至平面上（该平面垂直于 **a**）并计算最终结果，具体过程如下所示：

$$\mathbf{i}_N = \mathbf{i} - (\mathbf{a}\cdot\mathbf{i})\,\mathbf{a}$$

$$\mathbf{v}_N = \mathbf{v} - (\mathbf{a}\cdot\mathbf{v})\,\mathbf{a}$$

$$\begin{aligned}\mathbf{i}_N\cdot\mathbf{v}_N &= (\mathbf{i}-(\mathbf{a}\cdot\mathbf{i})\,\mathbf{a})\cdot(\mathbf{v}-(\mathbf{a}\cdot\mathbf{v})\,\mathbf{a})\\ &= \mathbf{i}\cdot\mathbf{v} - (\mathbf{a}\cdot\mathbf{i})\,(\mathbf{a}\cdot\mathbf{v})\,(2-\mathbf{a}\cdot\mathbf{a})\end{aligned}$$

【提示】上式仅适用于 **i**（以及 **v**）与 **a** 不平行时；否则，需要使用不同的基向量加以比较。

这里，假设 **a** 为单位向量（否则，可对其执行标准化计算），根据旋转定义，**i** 和 **v** 的点积结果须相等，如下所示：

$$\cos\alpha = \mathbf{i}\cdot\mathbf{v} - (\mathbf{a}\cdot\mathbf{i})^2$$

上述处理过程可将两个或多个转换整合为单一的转换操作。相反，由于可将旋转 **R** 分解为多个围绕基向量的旋转行为，因而旋转行为通常可通过旋转向量的形式加以描述，并涵盖围绕 3 个轴向的角度值，在后续内容中还将多次提及到这一描述方法。

尽管上述数学符号工作良好，但最终依然会导致问题的出现。例如，如何表达平移行为？完整的转换也可在某一独立步骤中完成，并涵盖平移操作。否则，转换组合首先需要平移至原点处，执行旋转和缩放，再次加上位置数据并于随后再次执行平移操作。

18.2.2 转换矩阵

该问题的解决方法是使用第 17 章引入的齐次坐标，在第 4 个维度的帮助下，可通过 4×4 矩阵执行平移操作，如下所示：

$$\begin{pmatrix}1&0&0&a\\0&1&0&b\\0&0&1&c\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix}=\begin{pmatrix}x+a\\y+b\\z+c\\1\end{pmatrix}$$

需要注意的是，最后一行中的计算结果并不重要，且主要起到记录作用。另外，若在某一方向向量上执行平移操作，而非位置向量，则 w 坐标为 0，该向量不会受到平移操作的影响。

如果将结果矩阵与缩放和旋转矩阵结合，最终的通用转换形式如下所示：

$$\mathbf{T} = \mathbf{P}\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x\mathbf{S}$$

其中，**P** 表示为平移操作，**S** 表示为下列缩放操作：

$$\begin{pmatrix}a&0&0&0\\0&b&0&0\\0&0&c&0\\0&0&0&1\end{pmatrix}$$

且

$$\mathbf{R}_z=\begin{pmatrix}\cos\theta & \sin\theta & 0 & 0\\ -\sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

类似情形也适用于 $\mathbf{R}_y$ 和 $\mathbf{R}_x$。通过 3D 矩阵替换 4× 4 矩阵左上角的 3×3 矩阵，即可简单地创建非平移矩阵。

由于最下方一行通常为(0 0 0 1)，因而 **T** 包含 12 个未知项。然而，并非全部矩阵均为有效转换，例如行列式为 0 的无效矩阵。不难发现，有效的转换共计包含 9 个自由度，分别对应于 3 个旋转角、3 个缩放因子以及 3 个平移值。

构成转换操作的各个矩阵均可执行逆置操作。对于平移，可用相反数替换各值；对于旋转，则可逆置其旋转角；对于缩放，则可通过倒数替换缩放因子。实际上，平移和旋转的逆置操作还可实施进一步的简化，即转置操作。此时，对应矩阵称作正交矩阵，并符合刚体对象的转换特征。这也意味着，读者可逆置全部转换行为，并以相反顺序执行，如下所示：

$$\mathbf{T}^{-1}=\mathbf{S}^{-1}\,\mathbf{R}_x^{\mathbf{T}}\mathbf{R}_y^{\mathbf{T}}\mathbf{R}_z^{\mathbf{T}}\mathbf{P}^{\mathbf{T}}$$

该处理过程较为直观，并可记录各个原始转换行为。否则，读者需要将转换分解为分量形式，抑或执行前述逆置操作。

若将转换分解为位置、旋转以及缩放信息，首先需要从矩阵最后一列中消除平移元素。若计算缩放以及旋转，且将 **S** 乘以基向量时，需要将基向量乘以缩放因子 a，b，c。例如 $\mathbf{RSi}=\mathbf{R}(a\mathbf{i})=a(\mathbf{Ri})$。另外，由于 **R** 为刚体旋转，因而 **Ri** 为单位向量。

进一步讲，**Ri** 表示为 **R** 的第一列，据此，a 须为 **RS** 第一列的量值，**R** 的第一列表示为该向量的标准化版本。鉴于 b 和 c 也存在类似情形，因而可将当前转换分解为 3 个基本转换操作。

尽管如此必要，但将 **R** 进一步分解为围绕 3 个轴向的旋转同样可行。对此，须计算 θ，ϕ，ξ 并满足下列算式：

$$\begin{pmatrix}\cos\xi & \sin\xi & 0\\ -\sin\xi & \cos\xi & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}\cos\phi & 0 & \sin\phi\\ 0 & 1 & 0\\ -\sin\phi & 0 & \cos\phi\end{pmatrix}\begin{pmatrix}1 & 0 & 0\\ 0 & \cos\theta & \sin\theta\\ 0 & -\sin\theta & \cos\theta\end{pmatrix}=\begin{pmatrix}x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\\ z_1 & z_2 & z_3\end{pmatrix}$$

由于当前转换并不涉及 w 坐标，因而可暂时忽略该项。

由于 z 旋转并不会影响到 z 坐标，x 旋转不会对 x 坐标产生任何影响，因而有 $z_1=-\sin\phi$，进而得到 ϕ 值。随后，可得到 $z_2=-\cos\phi\sin\theta$，进而计算 θ 和 $x_1=\cos\phi\cos\xi$，并可得到 ξ 值。相应地，方程组可能包含多个解，因而不同的旋转集经组合后可得到正确的答案。尽管该过程并不复杂，但往往会导致问题的出现，当执行插值转换时尤是如此。

上述方案使得验证系统可有效地防止舍入误差，这也是该方案的一个优点。鉴于转换矩阵的冗余性，若引入了较小的计算因子，则误差积累使得转换操作最终处于无效状态。通过检测某一转换及其对应的基本转换，并确保二者间处于一致状态，则可有效地避免上述问题。在某种程度上，这将减少组合转换至单一矩阵过程中的数据值，对此，主值仅出现于转换应用于场景中的多个对象时。此时，原验证过程不再必需，仅需对修正后的转换操作实施验证即可。

18.3 转换应用

转换操作可极大地简化计算过程，本小节将对若干示例予以考察。

18.3.1 利用转换操作构建运动行为

在实际操作过程中，最为简单的计算方式是暂时忽略全转换矩阵，并将精力集中于原始的基本转换中。总体而言，应包含某些基于转换 **T** 的节点，并将其移至对应处，而全转换仅出现于计算模型顶点的真实位置时。例如，单位立方体包含 8 个顶点，即$(\pm0.5\ \pm0.5\ \pm0.5\ 1)^T$，并可通过相应的转换操作将其移至任意位置，或调整其空间尺寸。待对应的转换计算完毕后，即可与各顶点执行乘法运算，进而确定立方体的最新位置。

相同处理过程也适用于方向向量。最终，沿立方体某一边的向量将不会受到任何影响。然而，此处需要注意法线向量。由于缩放操作并不会维持直线间的角度，因而若转换操作涵盖缩放因素，则转换前垂直于直线或平面的方向向量在转换后无法保证这种垂直性。

若 **n** 和 **v** 表示为彼此垂直的向量，且针对转换后的向量 **Tv**，期望获得垂直于该向量的向量，则须计算新的转换 $\mathbf{T}_n$，并满足 $\mathbf{T}_n\mathbf{n}\mathbf{T}\mathbf{v} = 0$。根据点积定义，由于 $\mathbf{n}\cdot\mathbf{v} = \mathbf{u}^T\mathbf{v}$，因而有$(\mathbf{T}_n\mathbf{n})^T(\mathbf{T}\mathbf{v}) = 0$。类似地，$\mathbf{n}^T\mathbf{T}_n^T\mathbf{T}\mathbf{v}=0$。根据定义，由于 **n** 和 **v** 彼此垂直，若随后对方程进行求解，则可得到 $\mathbf{T}_n = (\mathbf{T}^{-1})^T$，即 **T** 的逆转置。

再次强调，此类计算通常于场景后方且通过 3D 程序进行处理，但读者有必要理解其工作原理——当深入讨论碰撞检测时这一点体现得尤为明显。

针对转换的应用方式进而实现某一常规任务，下面考察箭头对象与某一点之间的对齐方式（该点通过特定位置向量表示）。这里，假设箭头模型的根位置（即恒等转换下的外观）如图 18.2 所示，并指向正 *z* 轴。

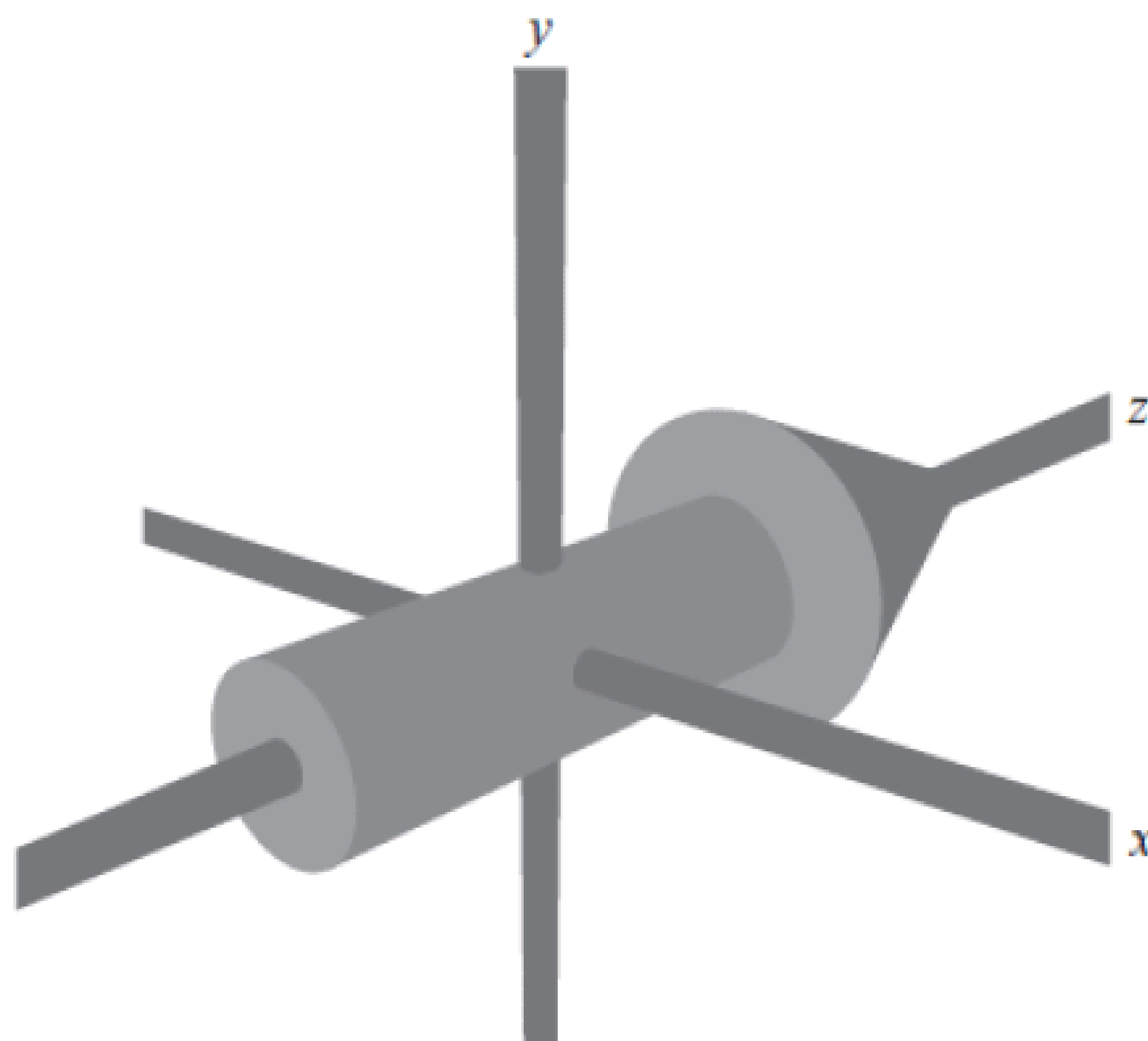


图 18.2　非转换状态下的箭头模型

这里，假设箭头位于位置 **p** 处，并指向向量 **v**。当计算 **v** 时，可首先对箭头对象执行转换操作。若已知该对象的转换操作，则 **v** 表示为经标准化后的转换矩阵中的第 3 列内容。随后，若箭头指向点 **q** 而非向量 **v**，则可通过适当的旋转加以实现。对此，可首先计算向量 **q** – **p**，即 z 轴转换后的新向量。经标准化计算后，该向量称作 **u**。通过叉积运算 **u**×**v**，即可得到箭头对象的旋转轴。根据点积运算 **u** · **v**，对应结果为旋转角的余弦值。

当前，全部工作即是执行旋转操作，围绕任意轴的通用旋转矩阵则稍显复杂，如下所示：

$$\mathbf{R}=\begin{pmatrix} c_+ + a_1^2c_- & a_1a_2c_- - a_3s & a_1a_3c_- + a_2s \\ a_1a_2c_- + a_3s & c_+ + a_2^2c_- & a_2a_3c_- - a_1s \\ a_1a_3c_- - a_2s & a_2a_3c_- + a_1s & c_+ + a_3^2c_- \end{pmatrix}$$

其中，对应轴表示为向量$(a_1\ a_2\ a_3)^{\mathrm{T}}$。若角度值表示为 θ，则 s，c_+，c_-分别定义为 $s = \sin\theta$，$c_+ = \cos\theta$，$c_- = 1 - c_+$。

针对非齐次坐标，则首先需要移至原点处，执行旋转操作并预算再次移回。由于平移操作通常最后执行，因而齐次矩阵的平移和旋转实际上彼此无关。最终，可生成转换 **S**，其中包含了左上角中的 **R** 以及单位位置数据。

据此，可将矩阵 **S** 与原始转换进行组合，进而应用旋转操作。在大多数情况下，该方案工作良好，但潜在问题依然存在。例如，针对转换行为，读者须判断执行右乘或左乘。为了解决这一问题，考察 **S** 于 **T** 后执行时的计算结果。此时，**S** 于左侧与 **T** 组合，即 **ST**，而非 **TS**。

【提示】相比较而言，可于先期执行非旋转状态下的对象的缩放操作，并于随后执行旋转操作。也就是说，若缩放行为先于 **T** 执行，则需要执行右乘计算。

若目标节点呈非对称状态，则结果又当如何？例如，若节点并非箭头模型而使用了相机模型，则后者无法整体指向某一特定位置。另外，还需额外指定向上方向，否则，相机将转向某一奇怪的方向。

为了处理这一类问题，读者须执行两次旋转操作。首先，可按照前述方式旋转对象，并指向局部 z 轴。随后，可再次围绕该方向向量旋转，并尽量保持与既定“向上”向量（通常为局部 y 轴）之间的一致状态，如图 18.3 所示。

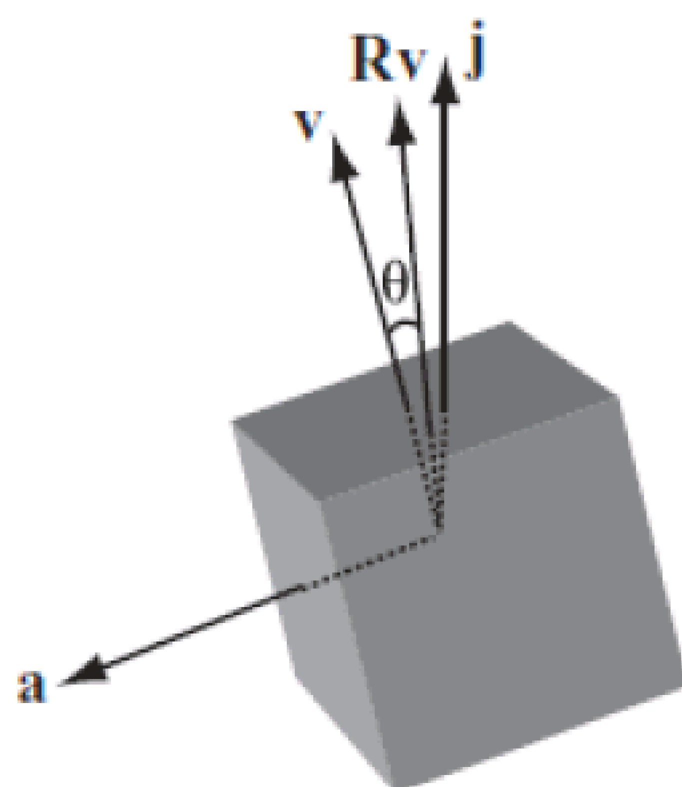


图 18.3　与既定“向上”向量保持一致

【提示】“向上”向量并无特别之处，在重力环境下，该向量通常可选取为垂直方向。

确定旋转角需要使用到微积分运算，即寻找满足 **Rv**·**j** 最大化的角度值。其中，**R** 表示为围

绕（标准化）指向 **a** = **q** – **p** 的旋转矩阵，**v** 则表示当前“向上”向量方向。代入矩阵 **R** 项后，可计算最大 θ 值，如下所示：

$$(a_1a_2v_1+a_2a_3v_3+a_2^2v_2)(1-\cos\theta)+v_2\cos\theta-(a_3v_1+a_1v_3)\sin\theta$$

经微分运算后，θ 值如下所示：

$$\tan\theta=\frac{(a_3v_1+a_1v_3)}{(a_1a_2v_1+a_2a_3v_3+(a_2^2-1)v_2)}$$

这将生成两个可能的 θ 值，且分别指向上、下方。另外，若分子和分母皆为 0，则方向向量 **a** 平行于 **j**，且全部方向与“向上”方向之间保持均等距离。

18.3.2　插值计算

两个矩阵之间的插值计算可生成平滑的运动效果，这一点在转换过程中十分重要。例如，设计人员开发一种可跟踪赛车的相机对象。相对于车辆，若将相机置于固定距离处，其效果相当于相机固定于车辆后方，而此处的实现目标是相机固定于直升飞机上，并尾随车辆运动。

对此，可计算两个转换行为。转换一为相机的当前转换 **C**（前述内容已对此有所讨论），转换二为目标转换 **T**，即相对于运动车辆的某一固定位置。此时，无须将相机移动至新位置。基于转换 t**C**+$(1-t)$**T**，只需沿其移动部分距离即可。

这一变化所产生的实际效果大大方便了相机的定位操作，当跟踪某一运动对象时（例如车辆），该方案工作良好。类似方法也适用于基于第三人称视角的游戏角色，例如游戏《超等马里奥世界》以及《古墓丽影》。其中，由于在运动过程中可能会遇到其他对象或地标建筑，因而计算目标转换相对复杂。总体而言，相机控制可视为游戏设计的难点之一，并对游戏体验产生显著的影响。

18.3.3　四元数

如前所述，可通过一类简单方法处理旋转问题，例如四元数。四元数表示为一类特定的 4D 向量，并可记为$(w\ \ x\ \ y\ \ z)^{\mathrm{T}}$。除此之外，四元数还可定义为 $w+xi+yj+zk$，其中，i，j，k 与虚数 i 相关。此处，虚数 i 表示为–1 的平方根。

这里，i，j，k 为正交虚数，且平方值均为–1 并满足下列各式：$ij=-ji=k$，$jk=-kj=i$，$ki=-ik=j$。同时，此类算式不适用于任何实数。虚数体现了一种数学抽象，并可生成相应的虚向量。

四元数 **q** 也可记为 w+**v** 形式，且 **v** 表示为向量$(x\ y\ z)^{\mathrm{T}}$。另外，还可定义 **q** 的共轭四元数，即 $w-$**v**。根据上述定义，两个四元数的乘积如下所示：

$$\begin{aligned}(w_1+x_1i+y_1j+z_1k)\,(w_2+x_2i+y_2j+z_2k) &= w_1\,(w_2+x_2i+y_2j+z_2k)\\ &+x_1i\,(w_2+x_2i+y_2j+z_2k)+y_1j\,(w_2+x_2i+y_2j+z_2k)+z_1k\,(w_2+x_2i+y_2j+z_2k)\\ &=(w_1w_2+w_1x_2i+w_1y_2j+w_1z_2k)+(x_1w_2i-x_1x_2+x_1y_2k+x_1z_2j)\\ &+(y_1w_2j-y_1x_2k-y_1y_2+y_1z_2i)+(z_1w_2k-z_1x_2j-z_1y_2i-z_1z_2)\\ &=(w_1w_2-x_1x_2-y_1y_2-z_1z_2)+(w_1x_2+x_1w_2+y_1z_2-z_1y_2)\,i\end{aligned}$$

$$+ (w_1y_2+y_1w_2-z_1x_2+x_1z_2)\,j+(w_1z_2+z_1w_2+x_1y_2-y_1x_2)\,k$$

若采用向量形式，则上述复杂内容可得到一定程度上的简化。若 $\mathbf{q}_1 = w_1+\mathbf{v}_1$ 且 $\mathbf{q}_2 = w_2+\mathbf{v}_2$，则二者乘积如下所示：

$$\mathbf{q}_1\mathbf{q}_2 = w_1w_2+\mathbf{v}_1\cdot\mathbf{v}_2+w_1\mathbf{v}_2+w_2\mathbf{v}_1+\mathbf{v}_1\times\mathbf{v}_2$$

需要注意的是，由于叉积分量，四元数乘法运算不包含交换律。

对此，四元数与其共轭四元数的乘积运算如下所示：

$$\mathrm{q}\overline{\mathbf{q}} = w_2-\mathbf{v}\cdot\mathbf{v}-\mathbf{v}\times\mathbf{v}$$

针对任意向量，由于 $\mathrm{v}\times\mathrm{v}=0$，因而有：

$$\mathrm{q}\overline{\mathbf{q}} = w_2-\mathbf{v}\cdot\mathbf{v}$$

由于 $i^2=j^2=k^2=-1$，因而可得到 $\mathbf{q}\overline{\mathbf{q}}=\mathbf{q}\cdot\mathbf{q}=|\mathbf{q}|^2$。四元数的逆可表示为 $\mathbf{q}^{-1}=\dfrac{\overline{\mathbf{q}}}{|\mathbf{q}|^2}$，特别地，针对单位四元数，则有 $\mathbf{q}^{-1}=\overline{\mathbf{q}}$。四元数与其逆四元数之间的乘积等于四元数$(1\ 0\ 0\ 0)^{\mathrm{T}}$，即标量 1。

对于旋转行为，若 **u** 表示为四元数（包含 0 标量），并针对四元数 **q** 计算乘积 $\mathbf{quq}^{-1}$，则可得到 **v** 围绕特定轴向的旋转结果。

注意，针对 **q** 的纯标量倍数，由于乘积 $\mathbf{quq}^{-1}$ 相同，因而可将 **q** 视为单位四元数 $\mathbf{s}+\mathbf{v}$，其逆四元数表示为$\overline{\mathbf{q}}=\mathbf{s}-\mathbf{v}$，这将生成下列乘积结果：

$$\begin{aligned}\mathbf{quq}^{-1} &= (-\mathbf{v}\cdot\mathbf{u}+s\mathbf{u}+\mathbf{v}\times\mathbf{u})\overline{\mathbf{q}}\\ &= -s\mathbf{v}\cdot\mathbf{u}+(s\mathbf{u}+\mathbf{v}\times\mathbf{u})\cdot\mathbf{v}+s(s\mathbf{u}+\mathbf{v}\times\mathbf{u})+(\mathbf{v}\cdot\mathbf{u})\mathbf{v}-(s\mathbf{u}+\mathbf{v}\times\mathbf{u})\times\mathbf{v}\\ &= \mathbf{v}\cdot(\mathbf{v}\times\mathbf{u})+s^2\mathbf{u}+2s(\mathbf{v}\times\mathbf{u})+(\mathbf{v}\cdot\mathbf{u})\mathbf{v}-(\mathbf{v}\times\mathbf{u})\times\mathbf{v}\end{aligned}$$

根据点积和叉积之间的计算关系，式中第一项为 0，最后一项为$|\mathbf{v}|^2-\mathbf{u}(\mathbf{v}\cdot\mathbf{u})\mathbf{v}$，因而有：

$$\mathbf{quq}^{-1}=(s^2-|\mathbf{v}|^2)\mathbf{u}+2s(\mathbf{v}\times\mathbf{u})+2(\mathbf{v}\cdot\mathbf{u})\mathbf{v}$$

若令 $t=|\mathbf{v}|$且 **a** 为标准化向量$\dfrac{\mathbf{v}}{t}$，则上式可重写为：

$$\mathbf{quq}^{-1}=(s^2-t^2)\mathbf{u}+2st(\mathbf{a}\times\mathbf{u})+2t^2(\mathbf{a}\cdot\mathbf{u})\mathbf{a}$$

这表示为围绕轴 **a** 的旋转行为，其中，角度 θ 可通过 $\cos\theta-s^2=t^2$，$\sin\theta=2st$，$1-\cos\theta=2t^2$ 加以确定。其中，仅需使用最后一个等式即可，前两个等式可推导出 $1-\cos\theta=2t^2$（**q** 包含单位长度）。最后一式等价于 $s^2+t^2=1$，即 $s=\cos\left(\dfrac{\theta}{2}\right)$。综上所述，围绕轴 **a** 且角度为 θ 的旋转行为可通过下列四元数加以描述：

$$\mathbf{q}=\cos\left(\frac{\theta}{2}\right)+\sin\left(\frac{\theta}{2}\right)\mathbf{a}$$

这可视为对矩阵形式的重大改进，其中，转换过程中涉及较少的计算。另外，与矩阵形式相比，旋转和四元数之间的关系更加直观。除此之外，还可通过线性组合在两个四元数之间执行插值计算。

18.3.4　父转换和子转换

除了不同位置间的组合转换（进而移动对象）之外，还可据此定义不同节点间的关系。通过

改变某一独立转换，则可移动全部节点组——调整某一节点的转换可影响模型网格的全部顶点。

相应地，可将 3D 场景世界中的全部节点实现树形排列。其中，各节点包含一个父节点和任意数量的子节点。最上方的节点为场景世界自身，而场景世界节点的子节点则包含了相对于前者的自身转换，但子节点的转换相对于其父节点而实施。换言之，若子节点的相对转换表示为 **C**，且父节点的转换为 **P**，则子节点相对于世界坐标系的转换可表示为 **CP**。

例如，假设某一模型对象位于场景世界中，且世界转换包含围绕 x 轴的旋转以及均匀缩放操作，该模型的任意子节点均在自身转换之前执行旋转和缩放操作。最终，模型及其子节点形成了一个组合，若父节点平移、旋转或缩放，则子节点（及其后续子孙节点）也将依此行事。

【提示】 在本节中，大多数平移操作也适用于旋转和缩放操作。

当转换某一对象时，此类运动存在潜在的不确定性。例如，对象沿 z 轴移动 5 个单位，此处的问题是，该轴可能为节点的局部 z 轴，或相对于父节点的 z 轴，抑或场景世界的 z 轴，如图 18.4 所示。

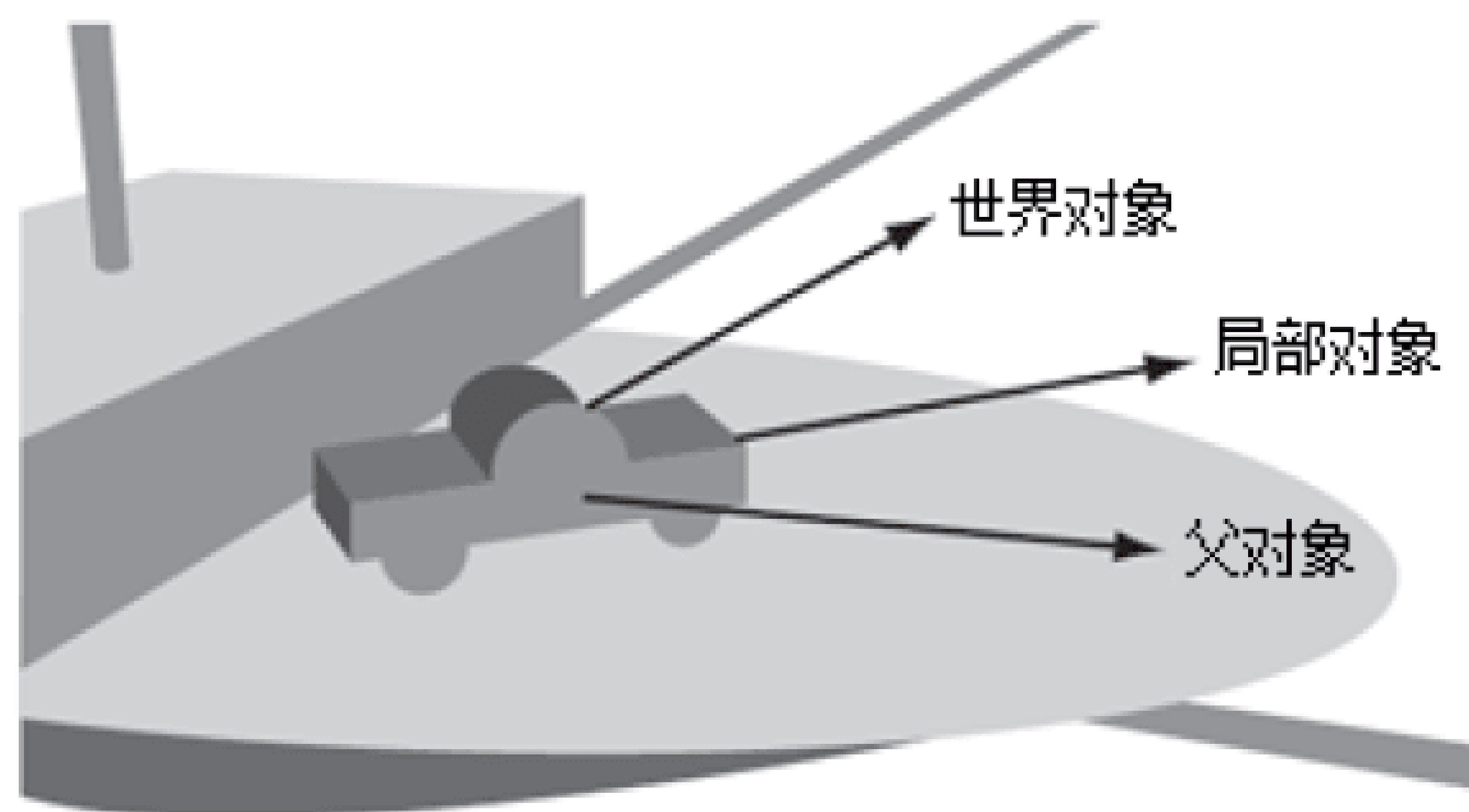


图 18.4　三维空间中的相对运动

图 18.4 显示了一个汽车模型，父节点为其上的船只模型。其中，车辆模型的 z 轴指向前向车身，若期望沿 z 轴平移车辆 5 个单位，则该行为包含多种含义。首先，车辆模型自身包含 z 轴，并指向世界向量$(1\ 0\ 1)^T$；其次是船体的 z 轴，并指向世界 x 轴；最后是世界 z 轴。因此，这涉及场景中的多个节点的 z 轴。

尽管如此，各节点的计算过程并不复杂。节点的 z 轴可通过世界转换矩阵的最后一列确定。通常情况下，任意方向向量均可通过与节点的世界转换得到。类似地，虽然计算父节点（或其他模型的）z 轴稍显繁琐，但世界 z 轴通常易于计算。

除此之外，其他方案则需要知晓节点本地转换的变化方式，进而使其沿特定向量运动。对此，局部参考坐标系中的对象转换可视为一类最为简单的例子，仅对象转换。

相对于父对象转换的转换调整工作稍显繁琐，为了理解其工作方式，假设相对于父 z 轴将对象移动 5 个像素。此处，首先需要计算子节点参考坐标系中的向量。若采用不同方式进行表达，则需要一个向量 $\mathbf{v}$ 并满足 $\mathbf{Cv} = \mathbf{k}$，因而有 $\mathbf{v} = \mathbf{C}^{-1}\mathbf{k}$。这意味着，当未转换至父空间中时，向量 $\mathbf{v}$ 等于 z 向量 $\mathbf{k}$。

需要注意的是，这与父节点自身转换无关。类似地，若相对于某一世界向量进行平移时，则需要逆置子节点的世界转换。

【提示】这里，向量 v 无须为单位向量，对应转换仅涉及缩放操作。实际上，这仅仅是出于方便考量。也就是说，可执行 5 个单位的平移操作且无须考察缩放行为。对此，可乘以已经缩放的向量 v。但对于旋转操作，则需要在执行旋转操作之前对轴向量执行标准化操作。

18.4 本 章 练 习

【练习 18.1】试编写函数以对相对于场景世界的特定节点、节点的父节点或节点自身实施平移操作、缩放操作和旋转操作。读者应思考使用何种方法存储转换信息，例如使用独立矩阵、3 个转换向量或旋转四元数。

18.5 本 章 小 结

本章讨论了齐次坐标以及某些数学概念（例如四元数）的应用方法，进而构建 3D 空间内节点的排列方式。期间，本章还进一步阐述了基于同步旋转、平移以及缩放转换的平滑运动（根据转换间的插值计算）的创建方法。

第 19 章将前述工作扩展至三维空间中，并对碰撞检测加以考察。

至此，读者应掌握如下内容：

- 术语转换的含义及其表达方式，其中包括：独立矩阵、矩阵组合以及三向量方式。
- 如何运用四元数创建简单的旋转表达方式。
- 转换的组合方式，并以此在不同位置间移动对象，以及相对于其他节点的固定位置构建对象组。

第 19 章　碰 撞 检 测

本章包含如下内容：

- 概述。
- 碰撞场景世界。
- 碰撞球体。
- 碰撞盒体。
- 碰撞柱体。
- 其他碰撞类型。
- 三维空间中的碰撞处理。

19.1　概　　述

与 2D 环境相比，3D 场景中的碰撞检测技术并无太多变化，当然，增加一个维度使得计算复杂度有所提升。首先，3D 环境中形状的种类有所增加，例如圆柱体和圆锥体。另外，两个对象间非碰撞状态的判断方式也趋于多元化，在 2D 环境中，两条彼此不平行的直线彼此相交；而在 3D 环境中，二者间仍可处于非相交状态。

尽管复杂度有所增加，但本章所探讨的大多数技术依然源自前述内容。由于本章主要阐述基础性的概念问题，因而不会涉及太多的代码。这里，假设读者可应用前述示例进而体现其数学内容。另外，本章的主要目标是创建一个工具集，并以此处理 3D 碰撞检测。除此之外，读者还将考察某些不同场合下的技术示例。

19.2　碰撞场景世界

球体间的碰撞可视为 3D 环境中最为简单的碰撞行为，其中，球体类似于 2D 环境中的圆。基于球体的大多数碰撞检测技术通常可视为应用于 2D 对象上的平移操作。

19.2.1　球体

球体可定义为与中心位置保持恒定距离 r 的点集。从数学角度上讲，圆可表示为一类特殊的球体。同时，球体可在任意维度上予以表示。例如，当维度大于 3 时，对应球体称作超球体。最

后，球体的数学表示法记为 S(c, r)。

与圆上一点相比，球体上一点的确定过程则相对复杂。当采用向量表示时，球体上的一点可定义为 $c+r\mathbf{v}$，其中，$\mathbf{v}$ 定义为单位向量。然而，此处并不存在明晰的三角关系以描述 $\mathbf{v}$。对此，可于先期考察一个单位向量，例如 $\mathbf{i}$，并于随后在两个方向上旋转该向量，这将生成此类向量的通用表达形式，如下所示：

$$\begin{aligned}\mathbf{v} &= \mathbf{R}_y\mathbf{R}_z\mathbf{i}\\ &= \begin{pmatrix}\cos\phi & 0 & -\sin\phi\\ 0 & 1 & 0\\ \sin\phi & 0 & \cos\phi\end{pmatrix}\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix}\\ &= \begin{pmatrix}\cos\phi & 0 & -\sin\phi\\ 0 & 1 & 0\\ \sin\phi & 0 & \cos\phi\end{pmatrix}\begin{pmatrix}\cos\theta\\ \sin\theta\\ 0\end{pmatrix}\\ &= \begin{pmatrix}\cos\phi & & \cos\theta\\ & \sin\theta & \\ \sin\phi & & \cos\theta\end{pmatrix}\end{aligned}$$

针对 θ、ϕ 以及球体点之间的一一映射关系，应在 0～2π 范围内取值，而 ϕ 应在 0～π 之间取值，并将其视为经纬度。当然，其他方案同样工作良好。

19.2.2　运动球体和墙面

如图 19.1 所示，相对于墙面或平面确定某一运动球体位置时，可将该过程转化为数学问题，即球体 S(c, r)沿向量 $\mathbf{v}$ 运动，且无限平面定义为点 $\mathbf{p}$ 和法向量 $\mathbf{n}$。据此，当前平面可表示为 P($\mathbf{p}$, $\mathbf{n}$)。

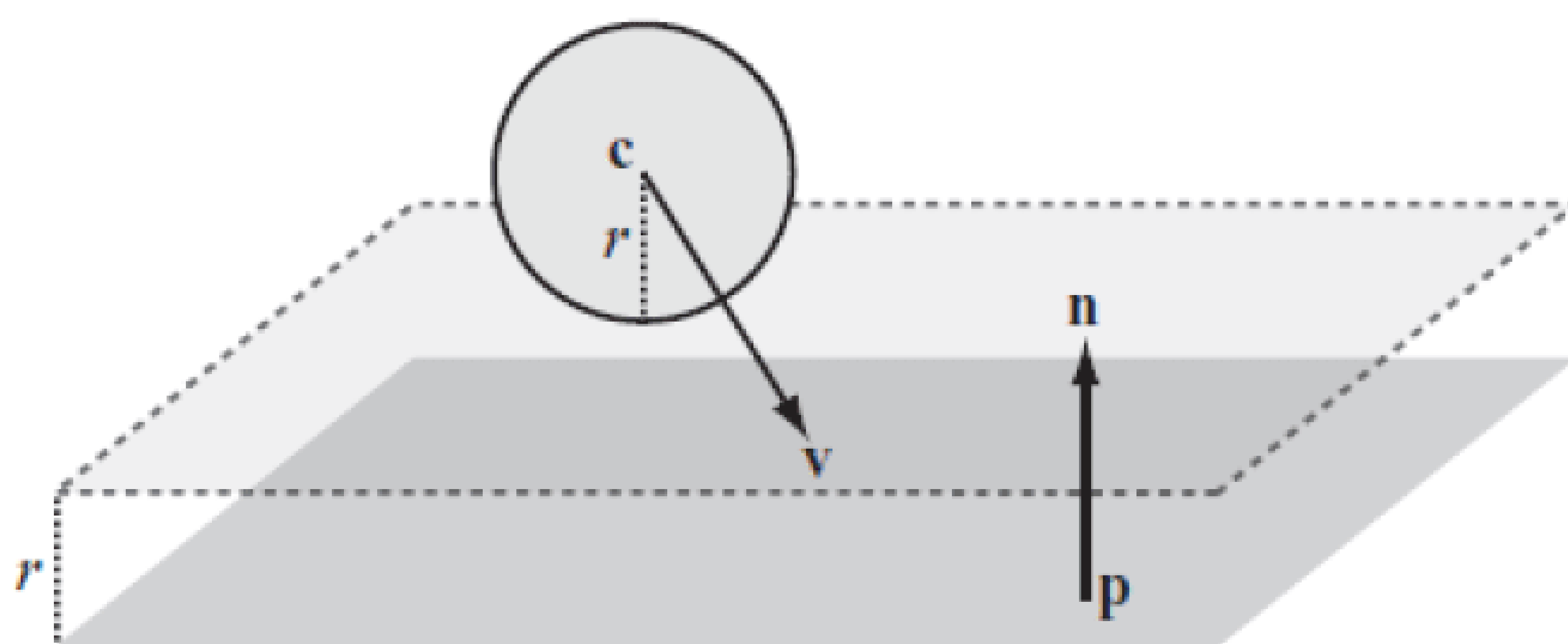


图 19.1　运动球体和平面

根据第 8 章中的结论，球体与平面之间的碰撞行为几乎等同于圆与直线间的碰撞过程，首先，可计算 $\mathbf{n}\cdot(\mathbf{c}-\mathbf{p})$值，若该值为正值，则球体位于平面的正法线一侧；否则，球体位于负法线一侧，此时，可将 $\mathbf{n}$ 替换为$-\mathbf{n}$。为了有效地简化当前问题，根据第 17 章中的相关结论，可计算点与平面之间的交点。对此，可通过向量 $r\mathbf{n}$ 偏移当前平面，进而计算直线 $c+t\mathbf{v}$ 与新平面 P($\mathbf{p}+r\mathbf{n}$,$\mathbf{n}$)之间的交点。

通常情况下，碰撞仅出现于平面的一侧，特别地，当计算实体之间的碰撞行为时，可令各表面的法线指向外侧，该条件可用于可见性判断中。若法线指向外侧，由于无须计算球体与平面之间的方位关系，因而球体与平面间的计算可在一定程度上得到简化。这里，仅当点积 **v** · **n** 为负值时，球体与平面产生碰撞。

19.2.3　球体和运动点或两个球体

下面考察球体与运动点或另一球体间的碰撞过程，其中，静止球体位于原点处，另一粒子或对象位于 **p** 处，且沿向量 **v** · **n** 运动。此处需要求解 **p**+*t***v**=*r*，该式等价于下列等式：

$$(\mathbf{p}+t\mathbf{v})\cdot(\mathbf{p}+t\mathbf{v})=r^2$$

$$\mathbf{p}\cdot\mathbf{p}+2t\mathbf{p}\cdot\mathbf{v}+2t^2\mathbf{v}\cdot\mathbf{v}=r^2$$

针对上述基于 t 的二次方程，经求解后可得到两个潜在的碰撞点。为了确定最终的有效碰撞点，需要将注意力集中于最小正值上；而负值则表明，粒子或对象位于原球体内部。

待解决了上述问题后，可简单地将其扩展为两个通用球体之间的碰撞。与两个圆形间的碰撞类似，两个半径为 r 和 s 的球体，其碰撞等同于粒子与半径为 r+s 的独立球体之间的碰撞。

19.2.4　碰撞点

当从 2D 移至 3D 环境后，也就是说，引进了额外的维度后，对象间不再仅是沿直线碰撞。相反，二者沿某一碰撞面相交。相应地，碰撞法线可视为碰撞检测过程中一类有效的工具。在碰撞处理过程中，仅碰撞的运动法线分量受到影响，切向分量不发生任何变化。

计算基于球体的碰撞法线相对直观，这一点与圆形的碰撞法线十分类似。其中，法线沿两中心位置形成的向量方向。当球体与平面发生碰撞时，该平面的法线即为碰撞法线。

19.3　碰 撞 球 体

待平面与球体间的碰撞处理完毕后，下面仅继续讨论不规则形状之间的碰撞行为，相关方法依然源自第 8 章，在本节中，2D 形状演变为 3D 形状。

19.3.1　椭球体

3D 椭球体对应于 2D 碰撞中的椭圆。类似于 2D 环境中圆与椭圆之间的关系，在 3D 环境中，一种较为简单的椭球考察方式是在三个维度上执行缩放操作。对此，针对位于原点的椭球体，其上一点的通用形式可表示为 **v** = **RSi**。实际上，若涵盖位置、方向和缩放操作，则可通过转换 T 准确地描述椭球体，并采用$(\cos\phi\ \ \cos\theta, \sin\theta, \sin\phi\ \ \cos\theta)^{\mathrm{T}}$应用于单位球体上。

【提示】沿某一轴向的、包含圆形横截面的椭球体称作回转椭球体。其中，椭球体可呈现为扁平状（类似于 UFO）或偏长状（类似于橄榄球）。

在本章中，术语“转换”并未体现应有的严格意义，并采用了 3×3 矩阵描述转换中的旋转和缩放部分。为了描述清晰，可采用符号 E(**p**,**T**)以将转换中的位置部分分离开来。在实际操作过程中，采用 4×4 全转换往往更为高效。另外，类似于 2D 椭圆，还可通过主轴及数值 *a*，*b*，*c* 列表方式描述椭球体，且二者具有相同的描述方式。

19.3.2　椭球体和运动点或平面

针对沿向量 **v** 运动的椭球体 E(**p**,**T**)与某一点或平面之间的碰撞，其最佳方式可通过转换空间予以实现，进而椭球体将演变为球体。当给定前述椭圆描述后，需要逆置转换 **T**。随后，可计算转换后的平面与单位球体之间的碰撞，该球体始于 $\mathbf{T}^{-1}\mathbf{p}$，且位移为 $\mathbf{T}^{-1}\mathbf{p}$。

当前问题涉及椭球体（或椭圆）表面处的法线计算。针对平面，转换后的法线 $\mathbf{T}^{-1}\mathbf{n}$ 并非是转换后的平面法线。相反，这里有必要使用逆转置矩阵，即 **T** 的转置。类似地，待转置完毕后，在世界空间中确定碰撞法线时，需要对碰撞法线执行逆转置计算$(\mathbf{T}^{-1})^{\mathrm{T}}$。此形式的示例代码如下：

```
function ellipsoidPlaneCollision(ell, pl)
  set inverseTransform to inverseMatrix(ell.matrix)
  set inverseTranspose to transpose(ell.matrix)
  set planePoint to matrixMultiply(inverseTransform,
    pl.refPoint-ell.pos)
  set circleVel to matrixMultiply(inverseTransform,ell.displacement- pl.displacement)
  set normal to matrixMultiply(inverseTranspose, plane.normal)
  set t to circlePlaneCollision(circlePos, 1,circleVel, planePoint, normal)
  return t
end function
```

19.3.3　两个椭球体

在 3D 版本的碰撞计算中，下一个步骤涉及两个椭球体之间的碰撞。正式地讲，该问题可视为如何计算两个椭球体的交点。根据 2D 环境中两个椭圆之间碰撞检测的计算难度可知，当前问题无法通过代数方式进行处理。尽管如此，代数方案依然提供了一个良好的开端。对此，可根据矩阵代数考察当前问题。这里，假设椭球体定义为 $\mathbf{E}_1(\mathbf{0},\mathbf{T}_1)$和 $\mathbf{E}_2(\mathbf{p},\mathbf{T}_2)$，且相对速度为 **v**。此处须计算两个单位向量和 $\mathbf{u}_1$ 和 $\mathbf{u}_2$，并满足 $\mathbf{T}_1\mathbf{u}_1 = \mathbf{T}_2\mathbf{u}_2 + \mathbf{p} + t\mathbf{v}$。若将 $\mathbf{T}_1$ 替换至等式的另一侧，则有 $\mathbf{u}_1 = \mathbf{T}_1^{-1}\mathbf{T}_2\mathbf{u}_2 + \mathbf{T}_1^{-1}p + t\mathbf{T}^{-1}\mathbf{v}$ 。

当前问题的求解公式包含单位向量 $\mathbf{u}_1$ 和 $\mathbf{u}_2$（各包含两个自由度）以及标量 *t* 等未知项，为了有效地降低求解范围，可增加一个辅助条件，即两个椭球体交于一点。因而在碰撞点处，法线间彼此平行。由于点 **Tu** 处椭球体的法线定义为$(\mathbf{T}^{-1})^{\mathrm{T}}\,\mathbf{u}$，因而有：

$$(\mathbf{T}_1^{-1})^{\mathrm{T}}\mathbf{u}_1 = (\mathbf{T}_2^{-1})^{\mathrm{T}}\mathbf{u}_2$$

逆置首个转换，则上述表达式如下所示：

$$\mathbf{u}_1 = \mathbf{T}_1^{\mathrm{T}}(\mathbf{T}_2^{-1})^{\mathrm{T}}\mathbf{u}_2$$

并可记为 $\mathbf{u}_1 = \mathbf{M}\mathbf{u}_2$。对此，可从首个等式中消除 $\mathbf{u}_1$，进而有：

$$\mathbf{M}\mathbf{u}_2 = \mathbf{T}_1^{-1}\mathbf{T}_2\mathbf{u}_2 + \mathbf{T}_1^{-1}\mathbf{p} + t\mathbf{T}^{-1}\mathbf{v}$$

由于 $\mathbf{u}_2$ 已知为单位向量，两个计算结果构成了 3 个独立方程组，并包含 3 个未知项。鉴于涵盖三角项以及线性项，因而方程无法通过代数方式求解。相反，此处可采用近似方案。进一步讲，若椭球体之间包含相同尺寸（具有相同转换 **T**）或标量倍数（即转换 **T** 和 $a\mathbf{T}$），则问题还可得到进一步简化。若采用上述两种条件之一，则可逆置转换矩阵，以使当前问题转换为两个球体之间的碰撞。该问题之前已得到解决，如下所示：

$$\mathbf{u}_1 - a\mathbf{u}_2 = \mathbf{T}^{-1}\mathbf{p} + t\mathbf{T}^{-1}\mathbf{v}$$

$$(\mathbf{T}^{-1})^{\mathrm{T}}(\mathbf{u}_1 \pm \mathbf{u}_2) = 0$$

第二个方程表明，法线之间处于平行状态，并可得到两个计算结果，即 $\mathbf{u}_1 = \mathbf{u}_2$ 或 $\mathbf{u}_1 = -\mathbf{u}_2$。针对首个结果，对应状态可描述为：一个椭球体位于另一个椭球体内部；而对第二个结果，则可直接进入求解过程。待减去首个方程后，则可得到如下算式：

$$(1 + a)\,\mathbf{u}_1 = \mathbf{T}^{-1}\mathbf{p} + t\mathbf{T}^{-1}\mathbf{v}$$

$$|\,\mathbf{T}^{-1}\mathbf{p} + t\mathbf{T}^{-1}\mathbf{v}\,| = 1 - a$$

其中，$\mathbf{u}_1$ 表示为单位向量。随后，当前方法的后续步骤则遵循两个球体间的碰撞处理方案。

19.4 碰 撞 盒 体

下面将讨论 3D 范畴内的另一个话题，即立方体。这里，立方体类似于 2D 矩形。然而，类似于矩形与其他凸多边形之间的关系，立方体的碰撞检测也涉及多种形状（此类形状涵盖由多条边连接的任意数量的数据面），例如正方体、金字塔形状、切割钻石形状或球体，此类形状称作凸多面体。

19.4.1 盒体

立方体包含 8 个顶点并由 12 条边连接，进而形成 6 个表面，例如砖块对象。这里，可将通用立方体视为标准单位正方体（该正方体位于原点处）经 **T** 转换后的结果，这一点与椭圆体十分类似。另外，立方体的 8 个顶点可视作 8 个不同的组合。据此，可简单地测试一点 P 是否位于立方体内部。此处，可计算 $\mathbf{T}^{-1}\mathbf{p}$ 进而确定各坐标的绝对值是否小于 0.5。

19.4.2 盒体和移动点

如前所述，针对顶点的绝对值是否小于 0.5，此处提供了一种直线（即 $\mathbf{p}+t\mathbf{v}$）与盒体（其转换为 **T**）之间的交点计算方法。该方案计算最小 t 值，因而判断 $\mathbf{T}^{-1}(\mathbf{p} + t\,\mathbf{v}) = \mathbf{T}^{-1}\mathbf{p} + t\mathbf{T}^{-1}\mathbf{v} + t\,\mathbf{w}$

是否位于正方体内部，该过程涉及 6 个不等式，如下所示：

$$-0.5<q_1+t\,w_1<0.5$$
$$-0.5<q_2+t\,w_2<0.5$$
$$-0.5<q_3+t\,w_3<0.5$$

针对求解线性不等式方程组，存在一类高效的方法可对其进行求解，即单纯型算法（simplex algorithm）。然而，该算法仅适用于立方体，其具体内容则超出了本书的讨论范围。相比之下，一类更通用的方法则是检测立方体各面的交点。与前述正方形处理方案类似，若采用面法线与碰撞向量之间的点积运算，则可忽略正方体其他侧面中的数据面。

前述内容曾对粒子与平面之间的交点有所提及，一类快速计算方案则是获取与无限平面（对应平面穿越对象表面）3 个可能的碰撞点。随后，可进一步检测此类数据点，进而判断位于表面矩形内部的点。其中，该判断过程适用于任意凸多边形。如图 19.2 所示，对应测试可获取点与多边形之间正确的位置关系。

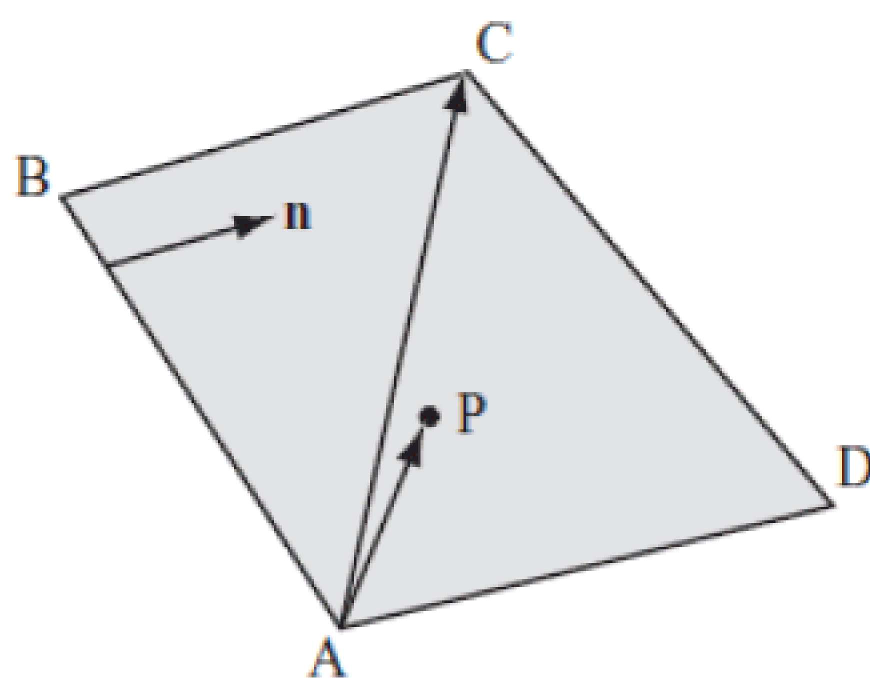


图 19.2　测试一点是否位于多边形内

在图 19.2 中，通过点积 $\overrightarrow{AP}\cdot\mathbf{n}$ 和 $\overrightarrow{AC}\cdot\mathbf{n}$，可测试点 P 是否位于 AB 的正确一侧。若二者包含相同的符号（即叉积值为正值），则 P 和 C 位于 AB 的同一侧。若针对 ABCD 各边均执行该测试，则可知 P 位于矩形内部。

相同的方法也适用于任意多面体。然而，若多面体表面非凸，则需要使用更为通用的测试技术，以确定点 P 是否位于对应表面上，例如第 10 章所讨论的光线跟踪技术。

19.4.3　两个盒体之间的碰撞

对于两个碰撞盒体，通常会采用面-面检测或更为特殊的面-顶点检测。当处理任意尺寸或角度的盒体时，可能会使用到多种面-顶点碰撞计算。其中，盒体的某一顶点可能与另一盒体的某一表面碰撞。也就是说，仅需检测立方体的前缘边处的顶点，而非全部顶点。实际上，除非立方体处于对齐状态，否则，针对各表面，仅存在单一顶点与其碰撞。图 19.3 显示了面法线方向上的、距当前面最近的顶点。

通过计算正方体各顶点 **v** 与另一正方体表面(**p**，**n**)之间的距离，可获得针对各表面的碰撞顶点，即(**v** − **p** · **n**)的最小值。若该距离为负值，则该表面不存在碰撞现象。同时，这一方法适用于任意凸多边形。若两个顶点距表面相等距离，则二者均可产生碰撞。

虽然通过计算距离可消除大部分工作，但依然需要对某些可能的碰撞行为进行检测。如

图 19.4 所示，该碰撞出现于两个盒体之间的边碰撞，同时也包括盒体间的边-面碰撞，且不存在顶点-面之间的碰撞，因而产生了一种完全不同的计算方法。

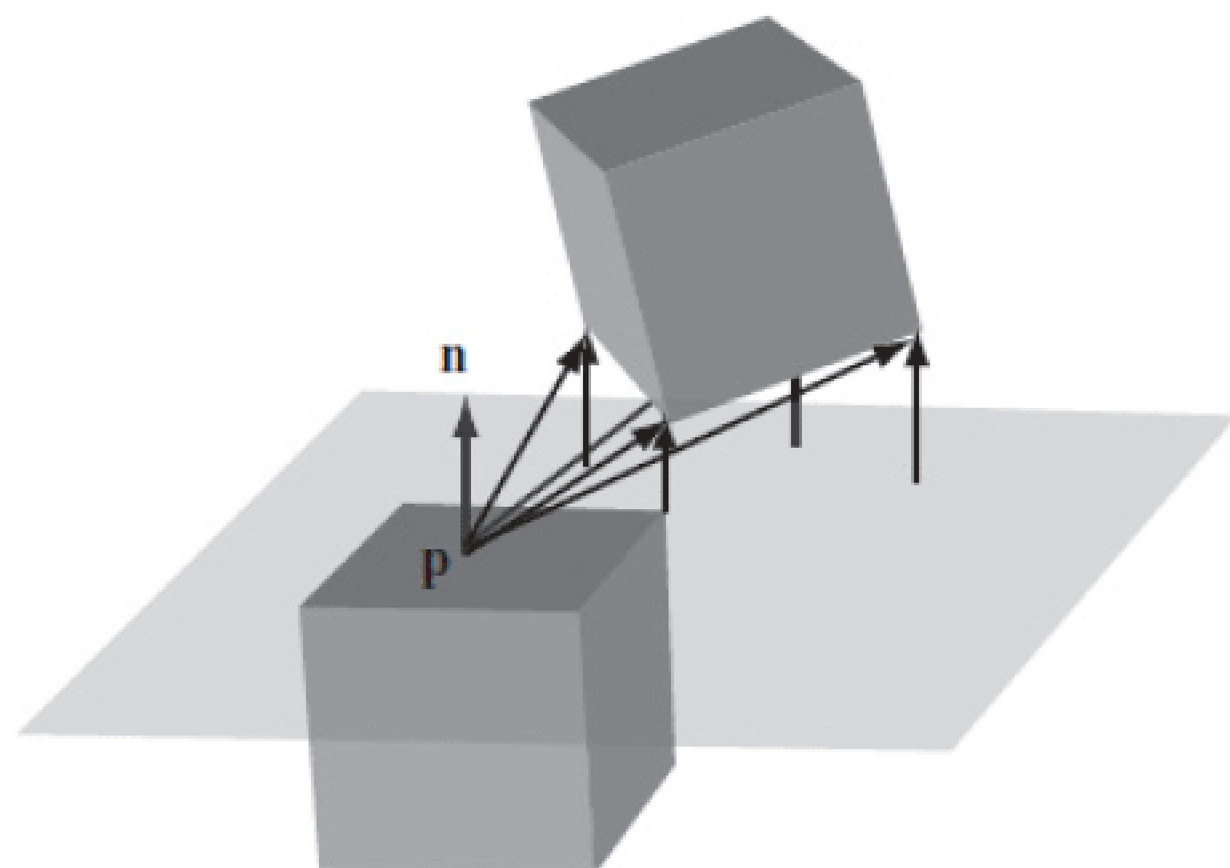


图 19.3　碰撞顶点

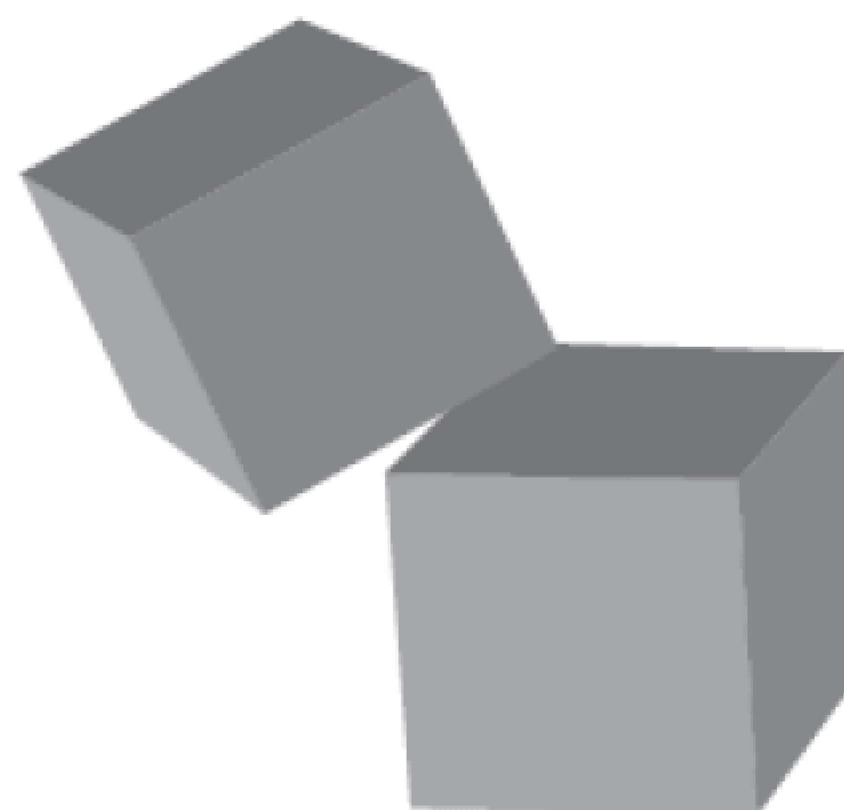


图 19.4　盒体之间的边-边碰撞

如图 19.5 所示，其中，平行四边形沿第一条边以及向量 **v** 方向扫掠。若平行四边形与第二条边相交，则碰撞产生于 c 处。

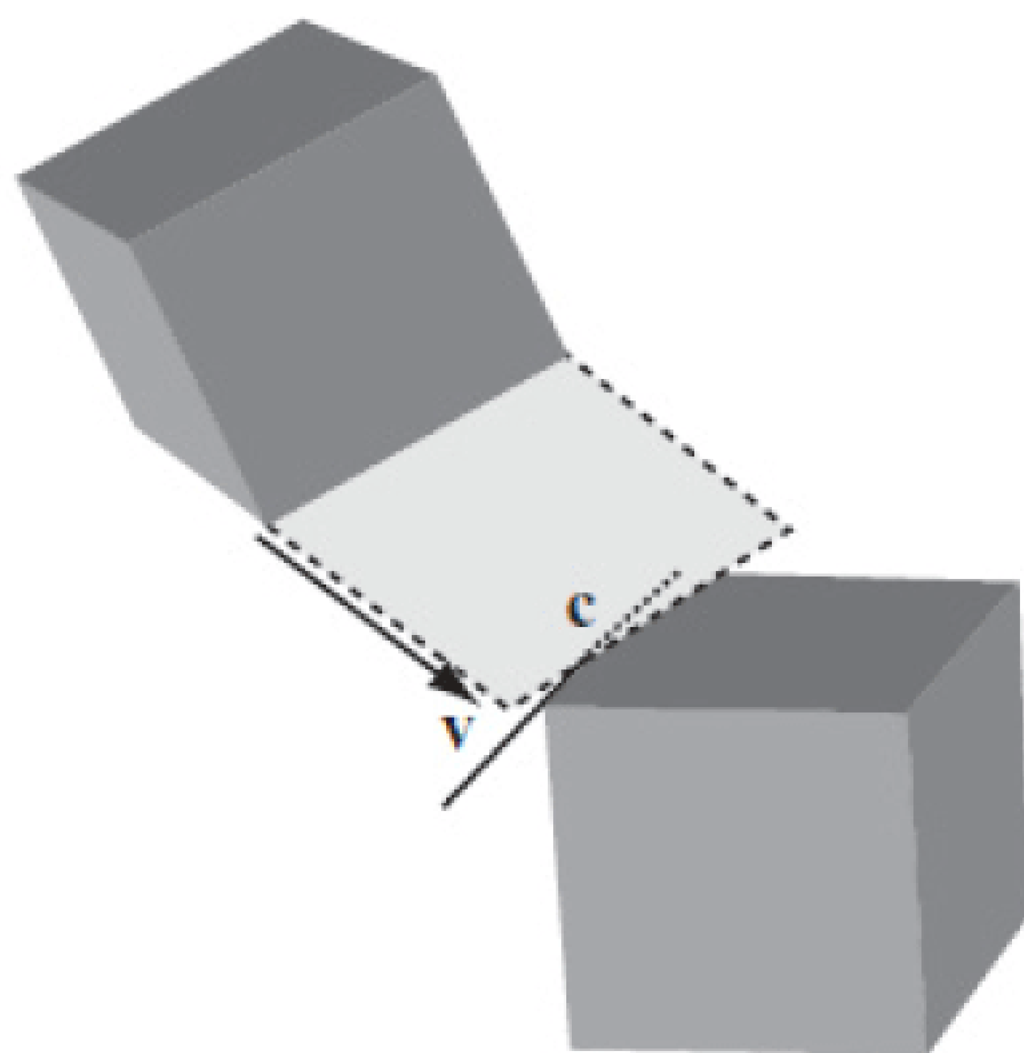


图 19.5　两条直线之间的碰撞

前述内容曾对图 19.5 所描述的碰撞类型有所讨论，即 4 个所涉表面与速度向量之间应得到正确的点积结果，并以此选取相应的碰撞结果。对于运动盒体，该向量为正向量，而对静止盒体而言，该向量则为负向量。类似于点-面碰撞，此处仅关注最近边。

【提示】第 23 章将具体讨论与碰撞相关的实现函数，并在轴对齐盒体和四边形（边与 *xz* 平面对齐）之间执行碰撞测试，该情形常见于 3D 游戏中。

19.4.4　盒体与球体之间的碰撞

球体可通过多种方式与盒体发生碰撞。例如，球体可与面、边或顶点产生碰撞。如前所述，基于顶点的碰撞可视为一类较为直观的点-球体碰撞，基于表面的碰撞处理过程也相对简单。对此，可首先计算球体与包含当前表面的无限平面之间是否产生碰撞。类似于多面体计算，随后可

检测碰撞点是否位于表面内部。

与顶点和面相比，边的碰撞检测则稍显复杂，这涉及球体和直线之间的碰撞计算，如图 19.6 所示。其中，球体 S(**p**, r)沿向量 **v** 运动，且直线表示为(**q**, **u**)。

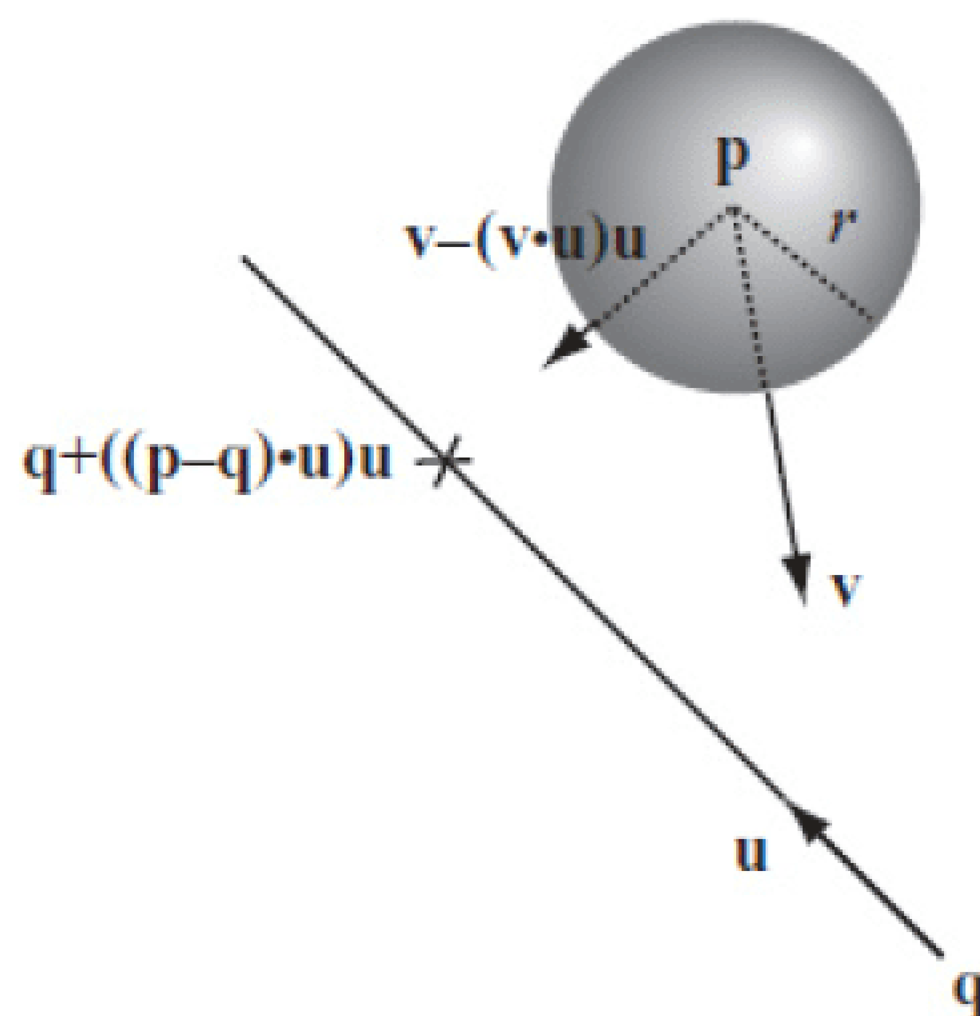


图 19.6　球体和直线

针对图 19.6 所示情景，一种简化方案是仅关注垂直于直线向量的运动分量。若从当前速度向量中减去(**v** · **u**)**u**，并向 **q** 加入向量((**p**–**q**) · **u**)**u**，则当前操作转换为 2D 空间内的投影问题。对此，可计算圆（圆心位于 **p** 处且半径为 r）与点 **q**+((**p**–**q**) · **u**)**u** 之间的交点。随后，可通过第 8 章或第 9 章中的方案求解当前问题。

为了减少计算量，可对球体-盒体碰撞进行适当的筛检。例如，仅当面法线与速度反向时，面-球体之间方有可能碰撞；仅当某一关联面法线与速度反向时，顶点-球体之间方有可能产生碰撞。

19.5　碰 撞 柱 体

最后一个形状则是圆柱体，且该形状不包含 2D 等价物。活塞以及某些饮水器具均可视为柱状体。

19.5.1　圆柱体

根据数学定义，圆柱体可视为与某一直线间距为 r 的点集，且该直线表示为圆柱体的轴向。圆柱体可表示为无限长的对象，或两端设置底面，其法线与轴向平行。通常，圆柱体的长度可定义为两个底面之间的距离，且无限圆柱体与垂直于轴向的任意平面之间的相交结果为一个圆。相反，针对非平行平面，相交结果为椭圆。对此，可采用 C(**p**, **v**, r, l)符号表示包含半径 r、轴向 **v**、长度 l 且中心位于 **p** 处的圆柱体。

圆柱体可视为圆锥体的特例，且二者存在相似之处，差别在于底面半径不同。如图 19.7 所

示，圆锥体中，圆横截面的半径正比于轴向距离。

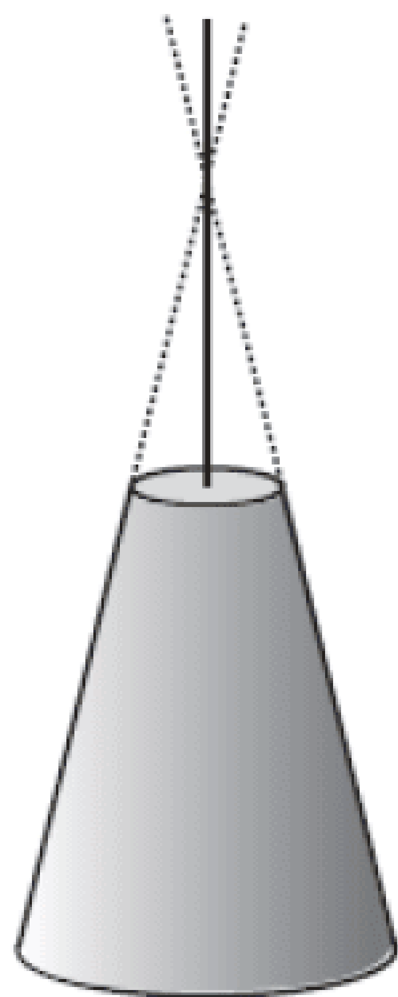


图 19.7　圆锥体

圆柱体和圆锥体均可视为旋转表面示例。作为 3D 形状，此类表面可通过 2D 轮廓并围绕某一轴向旋转而成，例如旋转转盘上陶壶的制作过程。大多数 3D 建模软件包均内置了旋转表面生成工具，在某些场合下，此类工具称作“车刀”。第 21 章将对旋转表面及其相关话题进行适当的扩展。

【提示】数学意义上的无限圆锥体类似于顶部连接的两个相等圆锥体，若采用一个平面切割此类圆锥体，对应结果将会得到 3 种形状：椭圆、抛物线或一种称作双曲线的对称形状。其中，双曲线可通过 $y^2 = 1 + x^2$ 加以描述。另外，上述 3 种形状总称为圆锥曲线。

19.5.2　圆柱体与点或球体之间的碰撞

本章前述内容实现了球体-球体以及球体-点碰撞过程中的大部分工作，而点与圆柱体之间的碰撞则等价于球体与直线间的碰撞行为（19.5.1 节对此有所介绍）。总体而言，半径为 r 的球体与半径为 s 的圆柱体之间的碰撞，其过程等价于半径为 $r + s$ 的球体与圆柱体轴向间的碰撞。

圆柱体的底面引入了额外的复杂度，如图 19.8 所示。对此，须计算与平面圆或半径为 r、圆心为 c 的圆之间的碰撞点。

关于图 19.8 所述问题，存在多种方案可对其进行求解，且全部方案具有等同效果。其中一种方法则从不同视角处理当前问题，并对平面（片状）对象十分有效。需要说明的是，对应方法与圆-直线间的碰撞关系紧密。据此，可于任意时刻计算球体与无限平面（该平面包含一个圆或其他片状对象）之间的相交结果。相应地，球体与平面的相交结果表示为一个圆，对应半径取决于球体与平面之间的距离。

为了表达相应的数学含义，此处令球体 S($\mathbf{p}$, r)沿向量 $\mathbf{v}$ 运动，并与 P($\mathbf{q}$, $\mathbf{n}$)平面相交。在时刻 t 处，球体与平面之间的距离定义为 $d=(\mathbf{p}-\mathbf{q}+t\mathbf{v})\cdot\mathbf{n}$，该值可为正值或负值。若$|d| > r$，则二者不相交；若二者相交，则相交结果表示为圆，该圆的半径为 $rc=\sqrt{r^2-d^2}$，圆心位置为 $\mathbf{pc}=\mathbf{p}+t\mathbf{v}-d\mathbf{n}$。

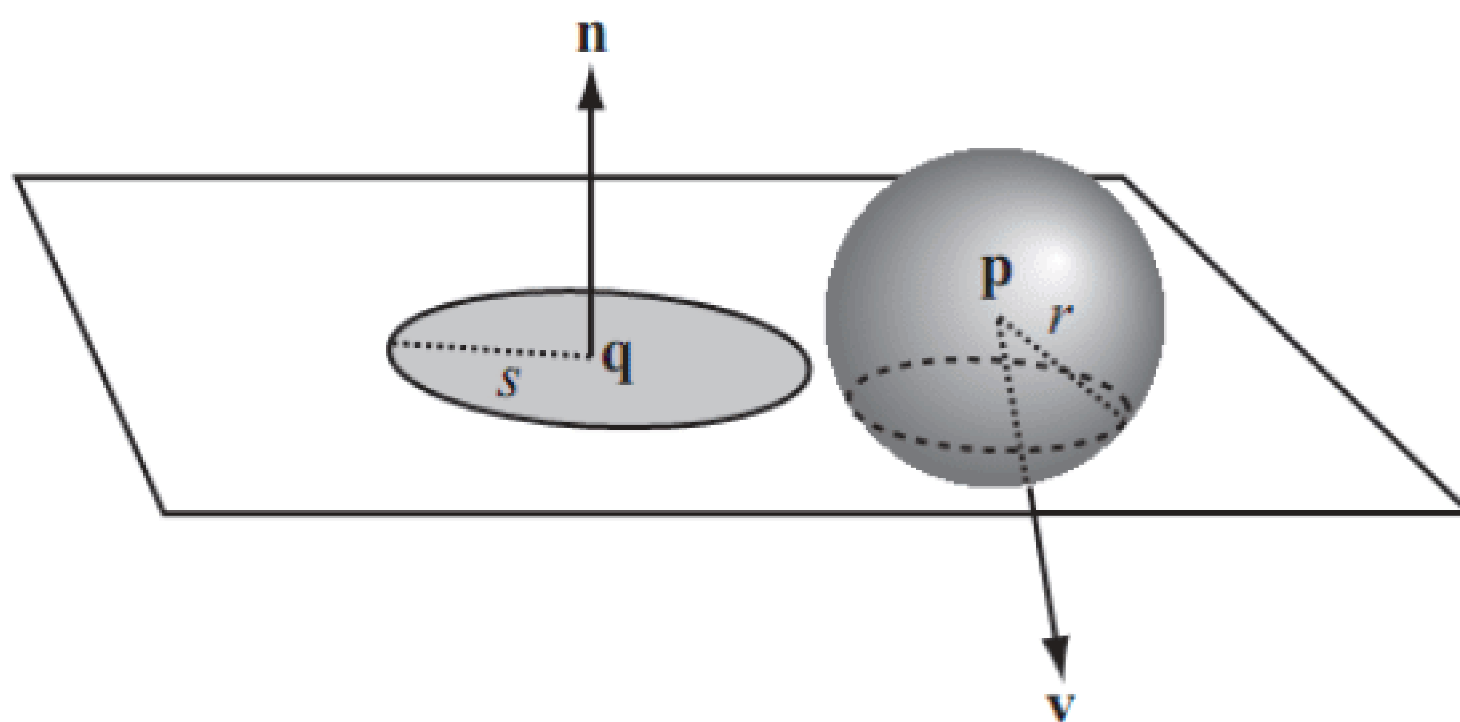


图 19.8　球体和平面圆

综上所述，下面计算球体与圆之间的碰撞结果。根据碰撞定义，该过程包含两个可能的碰撞结果。首先，若 d=r 且 $\mathbf{p}c$–$\mathbf{q}$≤s，则球体于圆形内部产生碰撞。对此，可得到下列算式：

$$(\mathbf{p}-\mathbf{q}+t\mathbf{v}-r\mathbf{n})\cdot(\mathbf{p}-\mathbf{q}+t\mathbf{v}-r\mathbf{n})\leqslant s^2$$

$$(\mathbf{p}-\mathbf{q}+t\mathbf{v})\cdot(\mathbf{p}-\mathbf{q}+t\mathbf{v})-2r(\mathbf{p}-\mathbf{q}+t\mathbf{v})\cdot\mathbf{n}+r^2\mathbf{n}\cdot\mathbf{n}\leqslant s^2$$

$$(\mathbf{p}-\mathbf{q}+t\mathbf{v})\cdot(\mathbf{p}-\mathbf{q}+t\mathbf{v})\leqslant s^2+r^2$$

若球体与片状平面之间存在碰撞点，则可据此进一步检测该点是否位于圆形内部。

【提示】读者可通过毕达哥拉斯定理获取相同的结果，此处仅在于彰显碰撞的逻辑，因而采用了相对冗长的计算方案。

第二种碰撞类型涉及球体和圆周间的碰撞，且体与片状平面的相交圆彼此相切。换而言之，其数学表达方式记为$|\mathbf{p}_c-\mathbf{q}|=r_c+s$。针对圆柱体，圆柱体仅与圆交于一侧且 d 为正值。

对于基于圆周的碰撞，须计算 t 值并满足下列条件：

$$(\mathbf{p}-\mathbf{q}+t\mathbf{v}-r\mathbf{n})\cdot(\mathbf{p}-\mathbf{q}+t\mathbf{v}-r\mathbf{n})=\left(\sqrt{r^2-d^2}+s\right)^2$$

$$(\mathbf{p}-\mathbf{q}+t\mathbf{v})\cdot(\mathbf{p}-\mathbf{q}+t\mathbf{v})-2rd=-d^2+s^2=2s\sqrt{r^2-d^2}$$

为了生成基于 t 的方程，可通过 d 的全点积展开式替换 d。这里的问题在于，结果函数无法通过代数方式求解。尽管如此，函数的输出结果依然较为平滑，实际上，该函数表示为二次函数。另外，对于函数自身，也可能存在无代数解这一情况。当从透视角度进行观察时，横截圆类似于处于运动状态下的椭圆，当然，该情形可通过修改参数进行调整。当处理 0 半径的粒子对象时，则可忽略圆周上的碰撞行为。

19.5.3　圆锥体与球体或粒子间的碰撞

当计算球体与圆锥体之间的碰撞时，须考察斜面的角度问题。类似于前述示例，可通过 r 扩展圆锥体，并将碰撞球体视为碰撞粒子。注意，还应对接触点予以谨慎处理。

出于简单考量，此处假设圆锥体的顶点位于原点处。其中，圆锥体的顶点处其半径为 0。若圆锥体无限长，则从顶点处，该对象以 α 角延展。若粒子位于 $\mathbf{p}+t\mathbf{v}$ 处，且圆锥体的标准化轴向量为 $\mathbf{u}$，则粒子与轴向之间的距离如下所示：

$$d=\sqrt{(\mathbf{p}+t\mathbf{v})\cdot(\mathbf{p}+t\mathbf{v})-((\mathbf{p}+t\mathbf{v})\cdot\mathbf{u})^2}$$

在时刻 t，包含粒子的平面其圆锥体半径为 $r = (\mathbf{p} + t\mathbf{v}) \cdot \mathbf{u}\tan\alpha$。

若粒子产生碰撞，则上述二值相等，进而可得到基于 t 的方程，如下所示：

$$\sqrt{(\mathbf{p}+t\mathbf{v})\cdot(\mathbf{p}+t\mathbf{v})-((\mathbf{p}+t\mathbf{v})\cdot\mathbf{u})^2} = (\mathbf{p}+t\mathbf{v})\cdot\mathbf{u}\tan\alpha$$

$$\begin{aligned}(\mathbf{p}+t\mathbf{v})\cdot(\mathbf{p}+t\mathbf{v}) &= ((\mathbf{p}+t\mathbf{v})\cdot\mathbf{u})^2(\tan^2\alpha+1)\\ &= ((\mathbf{p}+t\mathbf{v})\cdot\mathbf{u})^2/\cos\alpha\end{aligned}$$

不难发现，上式仅对圆柱体方程进行了微小改动，类似情形也出现于球体操作中，其差别仅在于底面的碰撞处理——在碰撞时刻，无须确保球体与底面间的距离为正值。

19.5.4　两个圆柱体间的碰撞

针对两个圆柱体之间的碰撞，可将其视为穿越某一平面的两个椭圆，该方案的碰撞结果沿圆柱体而非底面。对此，可在空间内选取某一平面，并将两个圆柱体投影至该平面上。若该平面垂直于轴向，则可适当简化计算过程。

另外，若圆柱体沿同一轴向对齐，当前问题则变得易于求解。随后，计算过程等价于处理两个处于运动状态的圆。相反，若圆柱体未与同一轴向对齐，则问题趋于复杂化。此时，需要处理椭圆间的碰撞，且应采用数值处理方案。

如图 19.9 所示，针对穿越圆柱体的某一平面，计算其上的投影椭圆需要得到该椭圆的中心位置。在该位置处，平面与轴向相交。这里，椭圆的半短轴表示为圆柱体的半径，并指向平面法线与当前轴线的叉积方向。通过半短轴与平面法线之间的叉积，则可进一步计算主轴向量。通过该向量与当前轴向间的夹角，利用三角计算则可获得长度值。

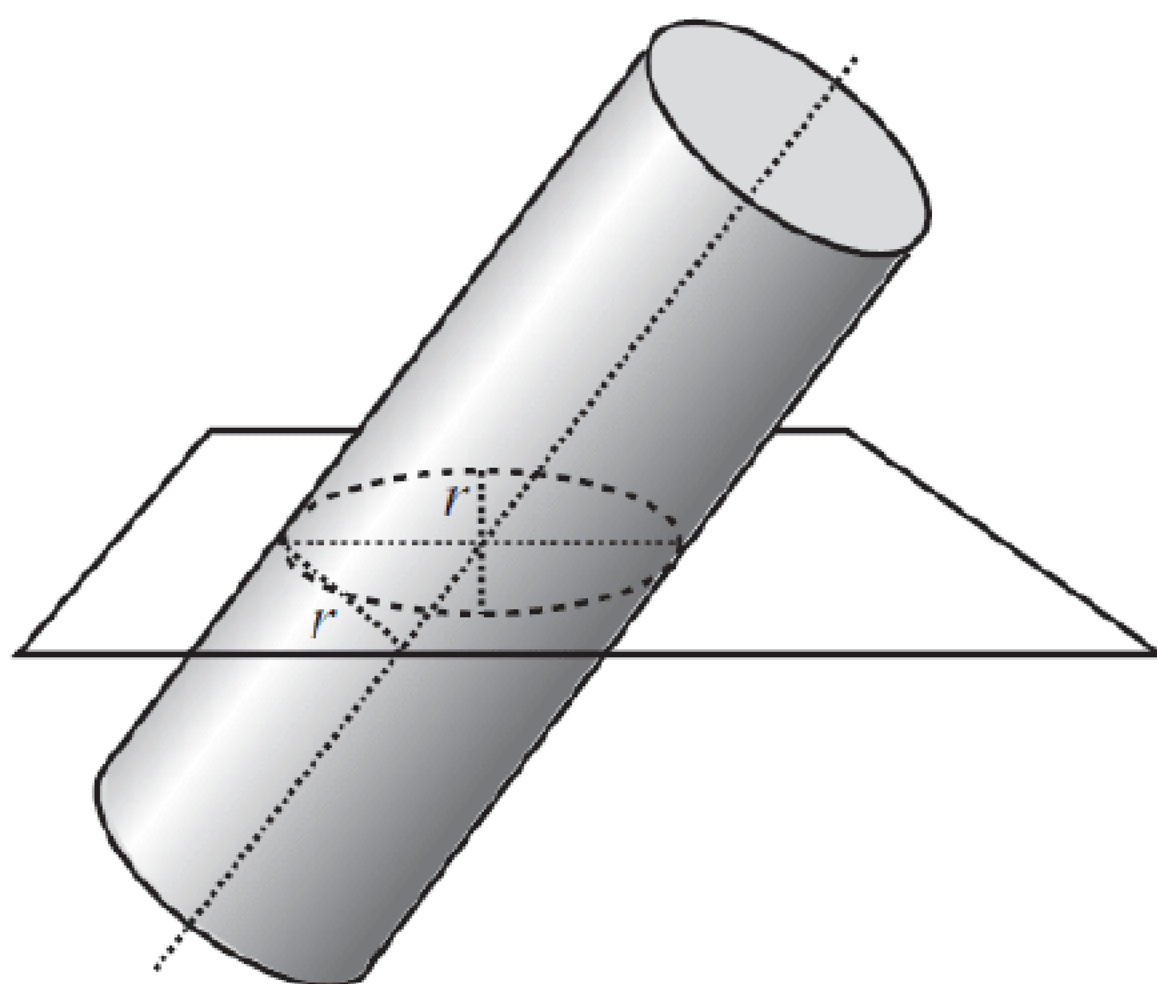


图 19.9　将圆柱体投影至平面上

19.6　其他碰撞类型

通过前述讲解可知，即使简单的形状也会涉及复杂的计算。另外，某些计算很可能无法得到有效的代数解。针对复杂的形状，应制定相关策略并在一定的时间要求下执行计算，这将涉及包

围球、包围椭球体以及包围盒。

19.6.1　包围球、包围椭球体与包围盒

类似于 2D 情形，复杂形状的碰撞计算涉及包围体的应用。顾名思义，包围体整体包含对象，若与实际形状较为接近，则可将包围体视为“代理”对象，进而执行碰撞检测计算。否则，包围形状可用于执行初始阶段的简单碰撞检测，并于随后进行逐三角形计算。

与 2D 场相比，3D 环境下的包围体计算并无太多变化。例如，可计算全部模型顶点的平均值以得到中心位置，进而计算半径以将其作为顶点值中心位置间的最大距离。该方案通常不会产生最小球体，但对应方法简单、快捷。如前所述，包围椭球体可通过因子分析进行计算，并可生成轴对齐或对象对齐的包围盒。

19.6.2　网格间的碰撞

针对复杂形状的碰撞检测，例如网格碰撞，通常缺乏较好的解决方案。这里，网格由大量的顶点构成，并通过三角形予以连接。其中，各三角形的法线视为已知数据并指向当前形状的外侧。双面网格则包含两个三角形集合，一组指向内侧，另一组指向外侧。由于法线方向常根据顶点的排列顺序进行计算，因而若三角形法线指向观察者，则顶点以顺时针方向排列。多数时候，若构建简化形状并将其作为代理对象，则上述过程可得到适当简化，进而计算较少的三角形。

实际上，三角形网格计算基本与盒体相同。例如，可采用图 19.2 所示技术确定点与三角形之间的位置关系，相应地，若点位于三角形平面内，且位于各边内侧，则该点位于网格内部。这也意味着，该点需与三角形其他顶点位于同一侧。因此，点积与（边和法线）叉积间的计算结果应为正值。也就是说，若 $\mathbf{n}_1=\mathbf{n}\times(\mathbf{v}_2-\mathbf{v}_3)$，则$(\mathbf{n}_1\cdot(\mathbf{v}_1-\mathbf{v}_2))\cdot(\mathbf{n}_1\cdot(\mathbf{p}-\mathbf{v}_2))\geqslant 0$。

19.7　三维空间中的碰撞处理

第 9 章和其他章节详细地讲述了 2D 碰撞处理方法。对于 3D 碰撞，其物理定律并无太多变化，相关对象以同一方式运行（包括弹跳和碰撞）。最终，对应技术也基本大同小异。需要注意的是，3D 计算将考察切平面，而非仅仅是切线。

这里，例外情况则是旋转操作。如前所述，侧旋可出现于多个方向上。而某一方向上的侧旋可认为是垂直方向上的 3 个侧旋。在特定环境下，有必要对此类现象予以考察，其基本概念并未发生太多变化，但这将使得计算过程趋于复杂化。

19.8　本 章 练 习

【练习 19.1】试将本章中的多个示例转化为实际函数，并定义相关函数以处理碰撞问题。其

中，大多数示例均未涉及实现代码且仅出现于 2D 碰撞中。相应地，球体和盒体的计算过程并不复杂。

19.9 本章小结

本章讨论了 3D 环境下的线性碰撞处理，且多数内容均源自 2D 碰撞检测。当前，读者已考察了大量的不同形状及其碰撞检测计算。除此之外，相信读者也对网格处理技术有所了解。当然，仍有许多细节内容需要进一步完善。第 20 章将讨论 3D 环境下的表面与光照之间的作用方式。

至此，读者应掌握如下内容：

- 球体、椭球体、回转椭球体、立方体、圆柱体、圆锥体以及网格的含义。
- 如何计算几何对象与光线之间的交点，以及与较小粒子间的碰撞。
- 如何计算球体与几何对象（以及类似对象组合）间的碰撞。
- 作为 2D 包围形状的扩展，如何通过包围体执行碰撞计算。

第 20 章　光照和纹理

本章包含如下内容：

- 概述。
- 光照。
- 材质。
- 着色机制。

20.1　概　　述

本章讨论对象与显示器之间的渲染方式，而非空间抽象实体的数学内容。其中，光照可视为核心内容。针对对象实体的创建，读者需要理解光照的本质及其实时模拟方式。

20.2　光　　照

在 3D 场景绘制于显示器之前，读者需要了解构成场景对象的各多边形的位置，以及绘制多边形所采用的颜色。相对于显示器，颜色可视为光照的简单应用，为了理解其原理，本节将引领读者快速考察光照的工作方式，以及基于复杂色彩效果的光照应用方式。

20.2.1　真实光照

当原子吸收并释放能量时，将以振荡电磁波的方式传递能量。取决于能量的大小，电磁波包含处于变化状态的频率和波长。人体中感官可感知某一范围内的频率，即光线。严格地讲，此类光线称作可见光。

人眼中的光线感知机制称作视杆细胞和视锥细胞，均各自具备不同的功能。其中，视杆细胞对亮度较为敏感，此处，亮度与不同的波幅关系密切。而视锥细胞则对频率更加敏感，因而其构造与视杆细胞相比稍显复杂。人眼中包含 3 种不同种类的视锥细胞且分别对应于不同的光线范围，对应范围分别是红、绿、蓝。

其他动物通常包含两种视锥细胞，因而观察到的色彩有限，这也是它们与人类的不同之处。然而，视锥细胞的不同仅是差异之一，其他动物可看到人类的视觉范围之外的光线。例如，蜜蜂可看到紫外线这一类高频光线。尽管不可见，但蛇类却能检测到红外线区域。红外线具有较低的

频率，且对于人类来说不可见。

尽管人眼中的视锥细胞可检测到红、绿、蓝光线，但可见色彩却远不止于此。大多数光线包含基于较大范围频率的多种叠加波。类似于视杆细胞，3 种视锥细胞对不同程度的光线均有所反应，多个频率的混合光线作为单一颜色被感知。另外，即使单一波长的光线无法准确地触发某一视锥细胞，但该光线仍可通过激活的邻接视锥细胞予以感知。

色彩源自辨识过程中的差异性，纯红和纯绿间的波长呈现为黄色；而绿色和蓝色间的波长则被感知为某种天蓝色，即青色；蓝色和黄色混合后则呈现为一类带有紫色的洋红色。总体而言，视觉系统按照红色-黄色-绿色-青色-蓝色-洋红色-红色这一循环顺序处理色彩问题，该循环自身则是生物进化的结果，且与光线波长无直接关系。同样，大量不同波长的混合结果则呈现为白色；相反，光线不存在之处将显示为黑色。

计算机工程利用了人类视觉系统特征并通过显示器展示颜色，其中，显示屏幕的各像素由红、绿、蓝（RGB）构成。对于高分辨率显示，各颜色值包含位于 0～255 之间的数值。当全部颜色均呈饱和之态，则显示效果为一个白点。相反，若全部颜色值为 0，则显示效果为黑点。取决于计算机的功率以及显示器的分辨率，可通过上述方式生成色彩斑斓的各种颜色。实际上，在真彩范围内，可生成 16777216 种不同的颜色。这里，真彩表示电子行业所沿用的一种规范，进而指明，色彩可通过基于红、绿、蓝的 256 种着色加以定义。

在线性代数中，可将各颜色值通过一个 3D 向量进行描述，该向量采用 0～1 之间的实数定义各颜色值数据。例如，颜色值<0 0.5 0.5>表示为中等强度的青色。因此，可对颜色执行数学运算，这也是该方案的优点之一。

大多数人仅将颜色视为光线的波长，实际上，这忽略了一个较为重要的问题。基于全部波长以及眼睛的感知结果，颜色主要源自大脑的意识判断。另外，颜色还受到周围环境的影响。例如，若场景中包含淡蓝色的环境光或强烈的阴影效果，则从感知角度来看，通常会减去这一全局值进而查看相关对象的基本颜色，大量的光学显示结果均使用了这一现象。

由于光线在物体表面反弹进而反射至观察者眼中，因而人们可看到场景中的各个对象。这里，物体表面通过不同方式与光线作用，例如，类似于弹性碰撞，镜面采用与入射相同的方式反射光线，即镜面反射。又如，白色球体吸收光线并在各方向上反射该光线，该现象称作漫反射或 Lambertian 反射。

黑色炭化表面吸收几乎全部光线，且不再向外界反射光线。相反，其热度上升并向外界辐射能量。红色表面则位于镜面和炭化表面之间，该表面吸收大部分光线，并以波长混合方式释放光线（在观感上为红色）。红色撞球包含两种表面，一种是经过特殊处理的釉面（无变化反射光线），另一种则是漫反射表面，进而吸收并反射光线。

在实时引擎中，上述因素在建模时将占用大量的计算时间，且需要通过某种技巧对其进行模拟，下面将讨论某些理想光源。

20.2.2 模拟光照

模拟光照（光源）是指作用于 3D 场景中渲染对象上的显示效果，并以此展现真实世界中的实际效果，因而分别存在环境光、有向光源以及衰减光源。相应地，光线于 3D 对象上的碰撞点

可视为上述 3 种光照效果的合成结果。图 20.1 显示了被不同光源照亮后的显示结果。

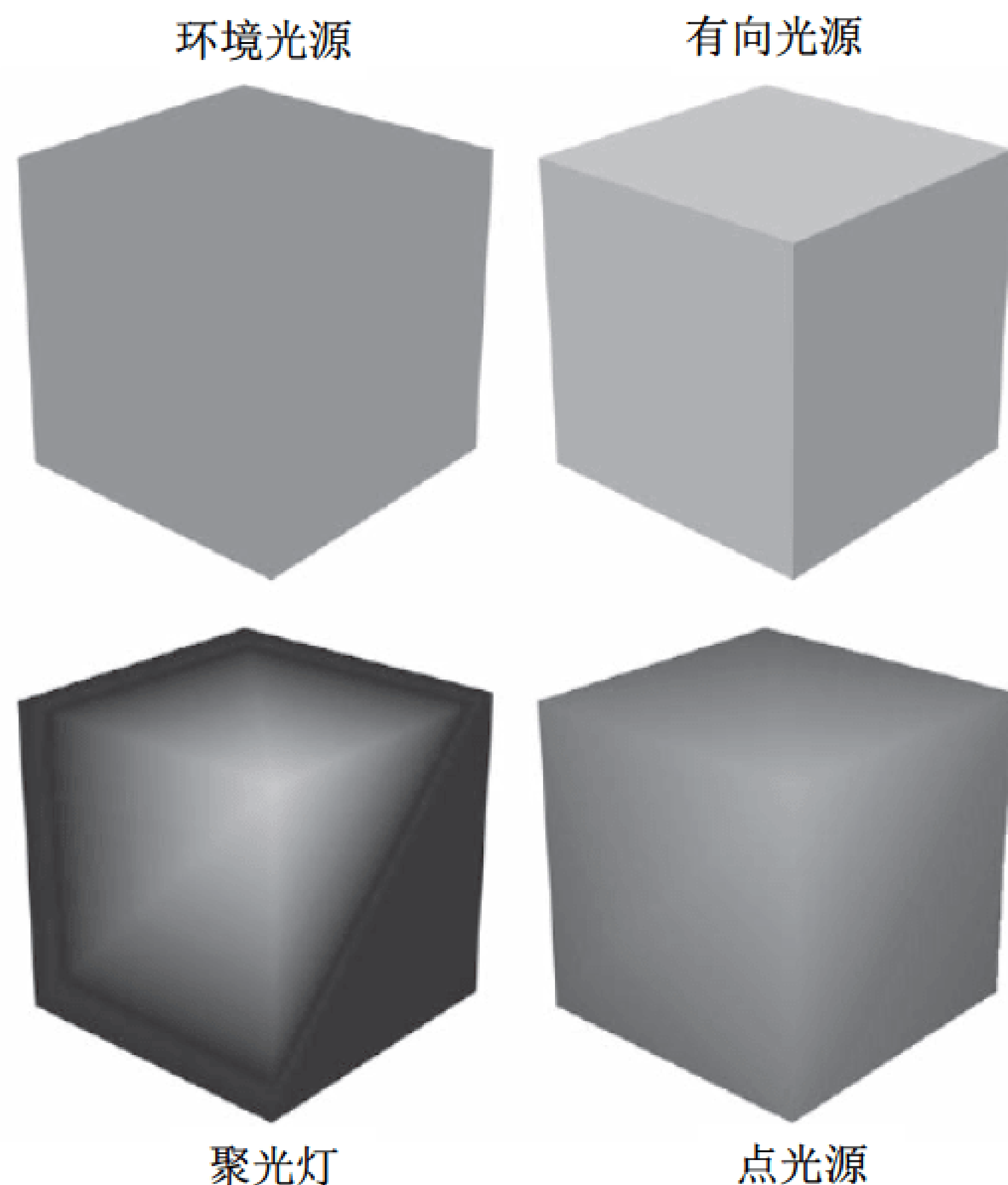


图 20.1　4 种不同的光照

其中，环境光模拟了观察者周围的光照，一种光照形式可描述为：光线从窗口射入，在房间内多次反弹，直至最终失去其光照方向。此时，环境光照亮了场景中的全部物体。在图形模拟过程中，鉴于光线通过上述方式反弹，因而其计算过程较为简单。实际上，环境光在各个方向上均等作用于多边形，且仅在某些场合下须对此进行适当调整。例如，复杂模型使得环境光的颜色和亮度在空间内产生变化。

顾名思义，有向光源源自某一特定方向，例如太阳，但并不会受到光源位置的影响。此类光源以均等方式照亮场景中的全部对象。另外，场景中可包含任意数量的有向光源。为了方便起见，有向光源常作为普通节点置于场景中，但仅方向向量与光照效果有关。

衰减光源包含两种形式，分别是聚光灯和点光源。对此，可将相关对象置于场景中，并赋予某一颜色值进而照亮附近对象。与远距离对象相比，与衰减光源较近的对象可得到更为明显的光照效果。该过程取决于 3 个衰减常数，并通过因子 b 调整光照的亮度。针对某一点光源，考察下列方程：

$$b=\frac{1}{k_1+k_2d+k_3d^2}$$

对于方向为 $\mathbf{u}$ 的聚光灯，在单位向量 $\mathbf{v}$ 处，且与表面间距离为 d 时，则有下列光照方程：

$$b=\frac{\max(-(\mathbf{u}\cdot\mathbf{v}),0)^p}{k_1+k_2d+k_3d^2}$$

其中，p 表示特定常数，用以表明光照的发散或聚焦程度。若 p 值较大，则光线呈现为狭长的光柱；而较小的 p 值可获得较大的光照范围。另一种方法是针对光线确定实际角度，并通过 $\max(-(\mathbf{u}\cdot\mathbf{v}),0)$项以确保指向光源的法线表面被照亮。

20.3 材　　质

各种形式的模拟光源均与场景对象表面作用，对应光线经反弹后使得对象可见。具体过程还与此表面特征有关。这里，表面可通过材质予以定义，并可将其视为作用于对象上的“涂层”。

20.3.1 表面颜色

材质体现了对象表面的不同属性，并涵盖不同的颜色分量。其中，数据值可通过独立值或图像贴图予以表达，20.3.2 节将对图像贴图进行深入讨论，当前可将其视为作用于整体表面的独立值。

自发光颜色可视为最简单的颜色元素，并表示为对象所发出的实际颜色，例如辉光灯。与真实的光源不同，自发光光源并不会对其他对象产生影响。当对场景进行模拟时，自发光光源通常易于计算，并提供了一种基于不同颜色的、简单的对象创建方式，对应颜色值表示为 C_{em}。

若对象表面被全光谱白光照射，漫反射颜色使得光线颜色具有 Lambertian 反射特征。换言之，此类光照可视为表面颜色的最佳候选光源。漫反射颜色与观察者的位置无关，但却与光线和表面间的角度关系紧密。如图 20.2 所示，对象表面被多条光线照亮，光线越靠近法线，则反射光线也就越强烈。

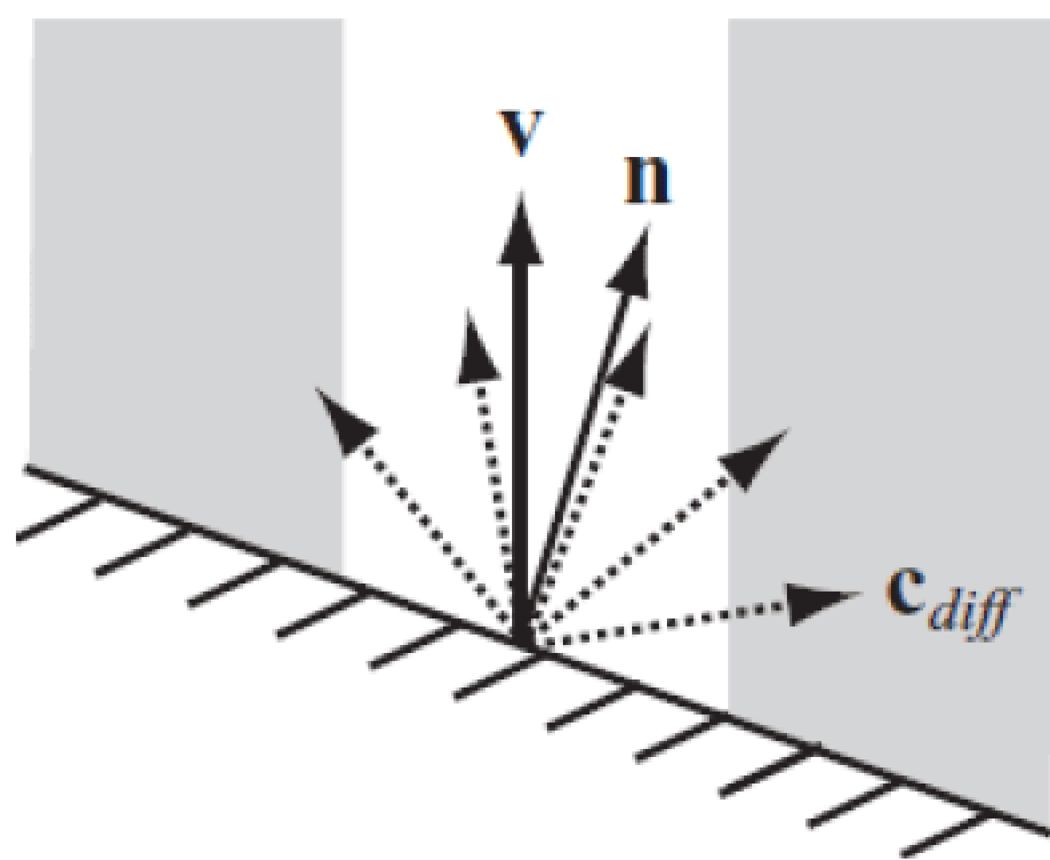


图 20.2　计算漫反射项

针对包含法线 **n** 的表面以及特定光源，可设定方程以计算漫反射项。若材质的漫反射颜色表示为 **d**，并被颜色为 c 的光源照亮（对应单位方向向量为 **v**），则 Lambertian 反射 c_{diff} 由 $\mathbf{cd}\max(\mathbf{u}\cdot\mathbf{v}, 0)$ 确定。其中，乘法运算采用分量形式执行。

针对 Lambertian 反射方程，需要注意的是，颜色向量不同于空间中的线性向量。除了其他方面之外，分量间的乘法运算在颜色间执行调制混合操作，例如，该操作使得蓝色向量趋向于红色。若对象表面吸收某一频率的光线，并反射其他光线，则上述方案工作良好。需要注意的是，由于环境光垂直于全部表面，因而其漫反射项仅为 **cd**。

镜面光包含两种元素，即颜色 s 和指数 m，经适当组合后，可产生单一颜色值，即镜面高光，但并不会生成镜面反射光。对于镜面反射光，除了基于空间各光源的、对象表面上的直接光照之

外，还应进一步考察其他对象反射的光线，即相对耗时的光线跟踪机制。据此限制，实时 3D 场景对象的光照行为仅受到光源自身的影响，而非其他对象所发射、吸收以及反射的光线。这一结果不仅影响到镜像关系，还会对实时阴影产生作用。

如前所述，镜面反射类似于弹性碰撞。如图 20.3 所示，光线在撞击对象表面后以相同角度反射。另外，视见向量越接近于（单位）反射向量 **r**，则镜面光变得越发明显。对此，可通过指数 m 调整向量间的接近程度，进而聚焦反射结果，这与通过指数 p 调整聚光灯十分类似。

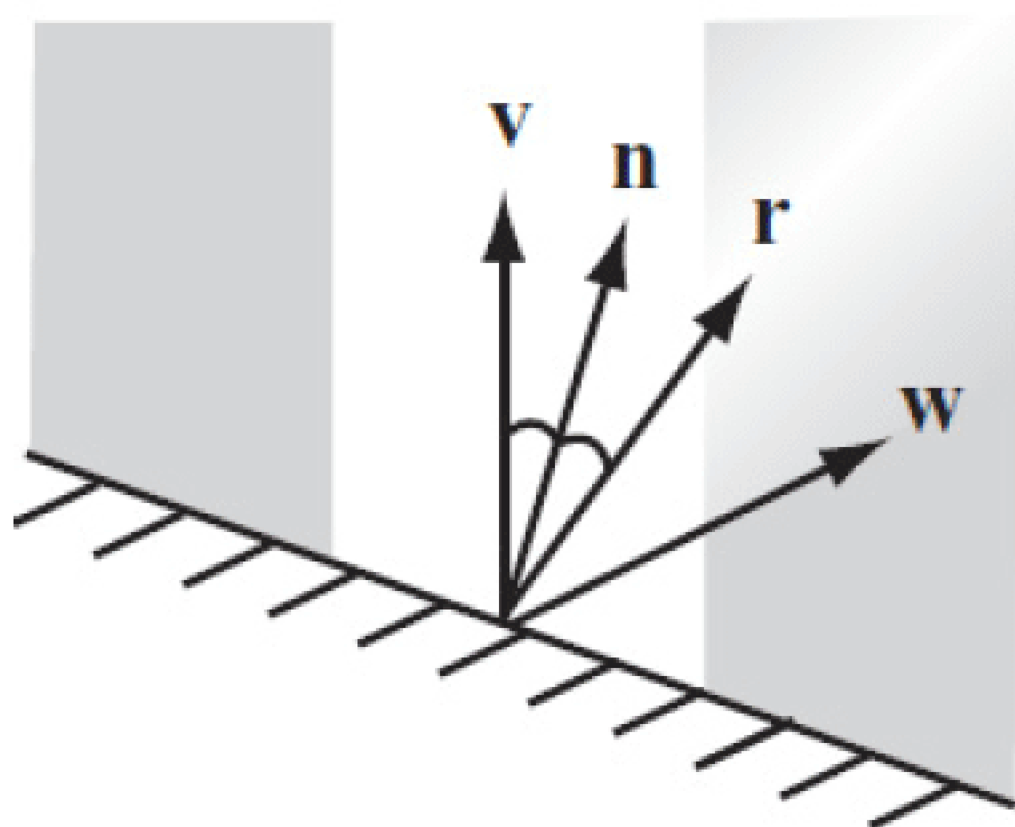


图 20.3　计算镜面光

这里，可根据特定的非环境光源并使用不同的方程计算镜面反射。对此，一类较好的方法假设 $\mathbf{n}\cdot\mathbf{v}>0$ 且观察者位于距表面点 **w** 向量处。待理解了这一关系后，则可得到 $\mathbf{c}_{spec}=\mathbf{s}\mathbf{c}\,(\max(\mathbf{r}\cdot\mathbf{w},0))^m$。此处，白色镜面色通常较为适宜，且指数 m 可设置为任意值——0 值将生成漫反射颜色，无穷大值在理论上可生成镜面表面。此时，仅沿 **r** 的视角可检测到光线。

针对非环境光源，上述各值须对各表面予以计算，场景中的各光源的结果值之和即为观察者看到的最终颜色值。为了生成该颜色值，可在各主颜色值上加上最大值 1。该方案旨在合成多个光源的颜色效果，且颜色间并非是调制关系。函数 surfaceColor()实现了上述理念，该函数的参数对应于前述光照效果，并辅以其他函数得以实现，如下所示：

```
function surfaceColor(normal, position,material, lights, observerPosition)
  set color to emissiveColor of material
  set observerVector to observerPosition - position

  repeat for each light in lights
    set lightColor to illumination(position, light)
    if light is not ambient then
      set v to the direction of light
      set diffuseAngle to max(-dotProd(normal, v),0)
      if diffuseAngle>0 then
        set diffuseComponent to modulate(diffuseColor of material,lightColor)
        add diffuseComponent*diffuseAngle to color

        set specularReflection to v + 2* dotProd(v, normal)
        set specularAngle tomax(dotProd(observerVector,specularReflection), 0)
        set brightness to power(specularAngle,specularFocus of material)

        set specularComponent tomodulate(specularColor of material, lightColor)
        add specularComponent*brightness to color
```

```
      end if
    otherwise
      add lightColor*diffuseColor of material to color
    end if
  end repeat
end function
```

其中，modulate()函数相对于向量值调制颜色值，如下所示：

```
function modulate(color1, color 2)
  return rgb(color1[1]*color2[1],color1[2]*color2[2], color1[3]*color2[3])
end function
```

illumination()函数所使用的参数则用于定义光源位置以及光源对象自身，如下所示：

```
function illumination(position, light)
  set color to the color of light
  if light is spot then
    set v to the position of light - position
    set brightnessAngle to max(-dotProd(v, direction of light), 0)
    if brightnessAngle=0 then return rgb(0,0,0)
    set brightness to power(brightnessAngle, angle factor of light)
    multiply color by brightness
  end if

  if light is spot or point then
    set d to mag(v)
    set denominator to the constant factor of light
    add the linear factor of light * d to denominator
    add the quadratic factor of light * d * d to denominator
    divide color by denominator
  end if
  return color
end function
```

材质还可包含附加数据，并通过图像贴图予以描述，稍后将对此进行介绍。

20.3.2 图像贴图和纹理

需要说明的是，并非全部对象均包含固定颜色，某些对象往往涵盖细节内容，进而向对象添加某种模式或纹理。对此，可使用一类称为“贴图”的图像。贴图包含了与几何形状表面相关的、传递至 3D API 的某些信息，且通常包含了较高的分辨率。稍后将讨论贴图的生成方式以及基于对象表面的投影方式，当前仅需了解贴图以逐像素方式定义表面参数值。进一步讲，当贴图应用于对象表面上时，由于图像贴图转换为纹素，因而相关操作将以逐纹素方式执行。通过这一方式以及具体的投影方法，贴图可作用于不同的表面区域。

下列内容展示了某些图像贴图示例：

- 纹理。纹理贴图有时也称作纹理，并用于调整表面的漫反射项。

- 光泽贴图。光泽贴图用于调整镜面项。
- 自发光贴图。自发光贴图用于调整自发光项。
- 光照贴图。光照贴图用于调整纹理贴图以及漫反射项。
- 反射贴图。反射贴图在主贴图上方生成反射图像。
- 凹凸贴图和法线贴图。凹凸贴图和法线贴图可生成更为复杂的表面外观。

【提示】实际上，反射贴图等同于纹理贴图，二者采用不同方式作用于对象表面上。

纹理贴图、光泽贴图以及自发光贴图通常易于理解，对应图像在特定点处生成材质的颜色值。通过定义图像与表面间的映射方式，可有效地改善最终的显示结果。另外，多个纹理经适当组合后可生成更为丰富的视觉效果，一如在 Adobe Photoshop 软件中，多个纹理可生成复杂的图像。这也是除了全局漫反射、镜面光以及自发光之外的另一个可作用于表面整体的可选项。

计算全部光照通常会消耗大量的处理器时间，多数时候，操作过程仅是重复计算某些相同值。另外，场景中的光源往往处于静止状态。据此，可预计算场景中的光照数据，并将其存储至纹理文件中，即烘焙机制。该机制可有效地降低实时光源数量，当然，这也将显著地增加纹理信息量。

为了解决这一问题，可创建第二个贴图，即光照图。光照图针对场景各部分定义了光照级别，其分辨率通常低于纹理贴图。通过光照图对纹理贴图进行调制，可在全部场景间复用纹理，并兼具图像贴图处理过程中的诸多优点，如图 20.4 所示。其中，第一幅图像中的纹理图与第二幅图像中的光照图合成（其分辨率较低），进而生成阴影图像。大多数 3D 建模软件均设置相关选项，进而可生成烘焙纹理，并可作为光照图或最新的完整纹理贴图。

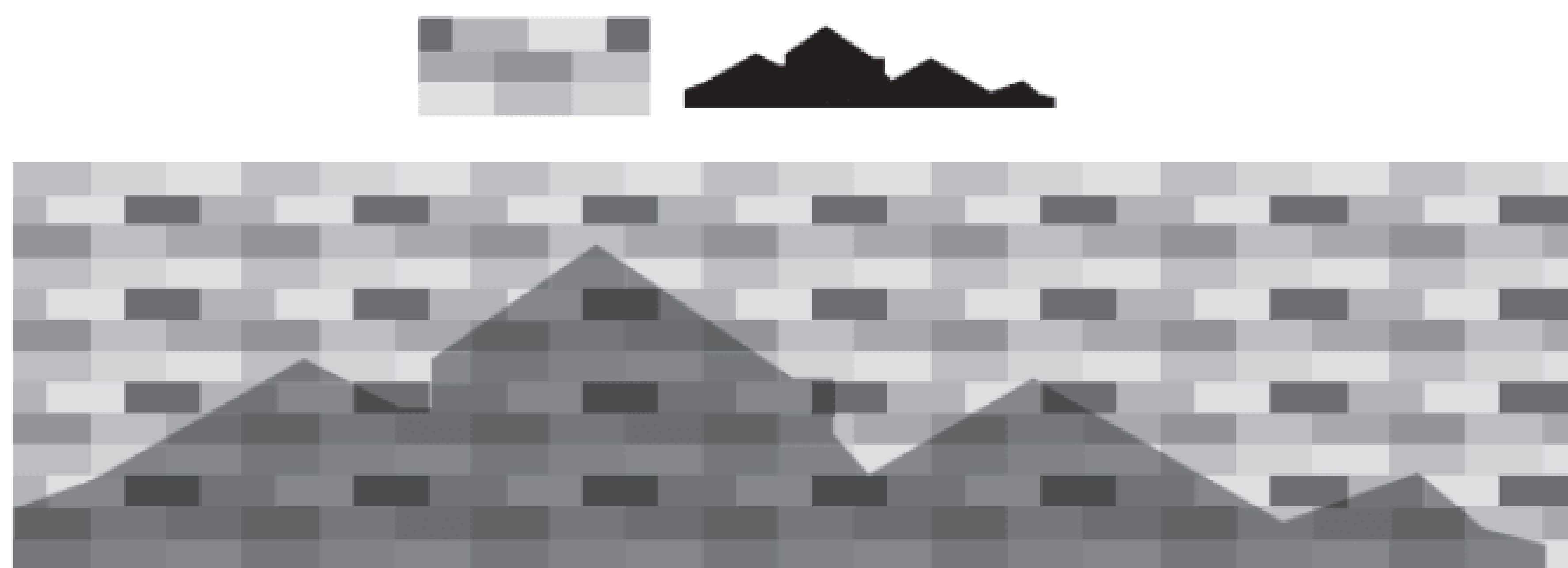


图 20.4　使用光照图调制纹理图

凹凸贴图可在小于多边形的细节级别上对高度变化进行建模，例如凹痕、气泡以及浮雕文字。此类效果可通过阴影和高光予以实现。需要说明的是，凹凸贴图并未改变对象的几何形状，该过程类似于法国错视画派笔下的 3D 图像。当观察者直视对象表面时，其效果尤为明显；然而，当水平观察图像时，其 3D 效果则大打折扣。

从本质上讲，凹凸贴图可视为高度图，图中各点体现了距表面间的距离。通常情况下，可采用灰度图生成此类效果。另一种替代方案是法线贴图，也就是说，可将（单位）向量的 x，y，z 坐标映射至红、绿、蓝数据值，以使各纹素编码为基于颜色数据的法线方向。最终，凹凸贴图等价于法线贴图。作为一种折中方案，法线贴图增加了内存占用空间，而凹凸贴图则在性能方面有所提升。如图 20.5 所示，法线贴图用于扰动表面法线，并以此改变方向和光源的镜面光线。

法线贴图源自凹凸贴图，并涉及高度图的读取方式。当使用高度图时，从本质上讲，可根据凹凸贴图的邻接数据比较像素的高度值。

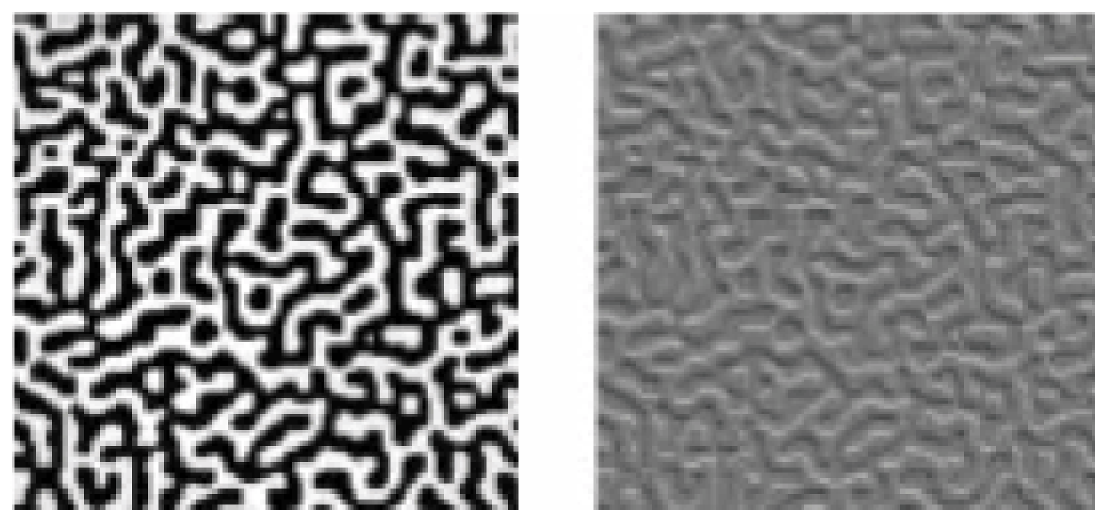


图 20.5　凹凸贴图在表面光照作用下的效果

20.3.3　贴图与形状之间的匹配

为了实现正确的图像贴图，须告知 3D 引擎图像与表面间的对应关系。对此，须创建图像与表面间的映射关系。换而言之，对于表面上的各点，需要将其与图像贴图中的纹素进行匹配。出于讨论目的，此处可将贴图视为纹理。

当在点与纹素间进行匹配时，可通过标准坐标对纹素进行标记。虽然某些材质会优先选取 3D 纹素（映射全部对象空间体，进而获得更为有趣的视觉效果），例如木纹状纹理，但纹理大多数时候表示为 2D 图像。为了描述清晰，可采用独立的坐标系对纹素进行标注，例如 s 和 t 或 u 和 v。

在纹素与表面的映射过程中，可采用适当的转换操作。实际上，此类操作较为有限且仅限于 2D 环境。然而，当执行缩放操作时，齐次坐标须保持不变，因而应在贴图绑定至多边形之前对其执行旋转、平移以及缩放操作。而对于映射过程，读者需要进一步了解其含义，下面列举了一些标准方案：

- 平面贴图。纹理整体穿越对象，并出现于另一侧。
- 圆柱体贴图。纹理以类似于纸筒的方式环绕对象。
- 球体贴图。纹理以球状方式分布于对象上。
- 立方体贴图。纹理经整合后形成一个立方体并映射至对象外部，该方案需要使用到 6 个纹理。
- 定制贴图。对于某些复杂网格，例如人物角色，需要针对网格中的各个三角形单独制定纹理坐标。

除了最后一个方法之外，其他方法提供了对象表面点 3D 信息与 2D 形式之间的转换手段。当采用平面贴图时，各顶点的 3D 坐标将投影至某一平面上，其 xy 坐标用于映射纹理贴图的 st 坐标。如图 20.6 所示，对象的 xy 坐标直接映射至图像的 st 坐标。

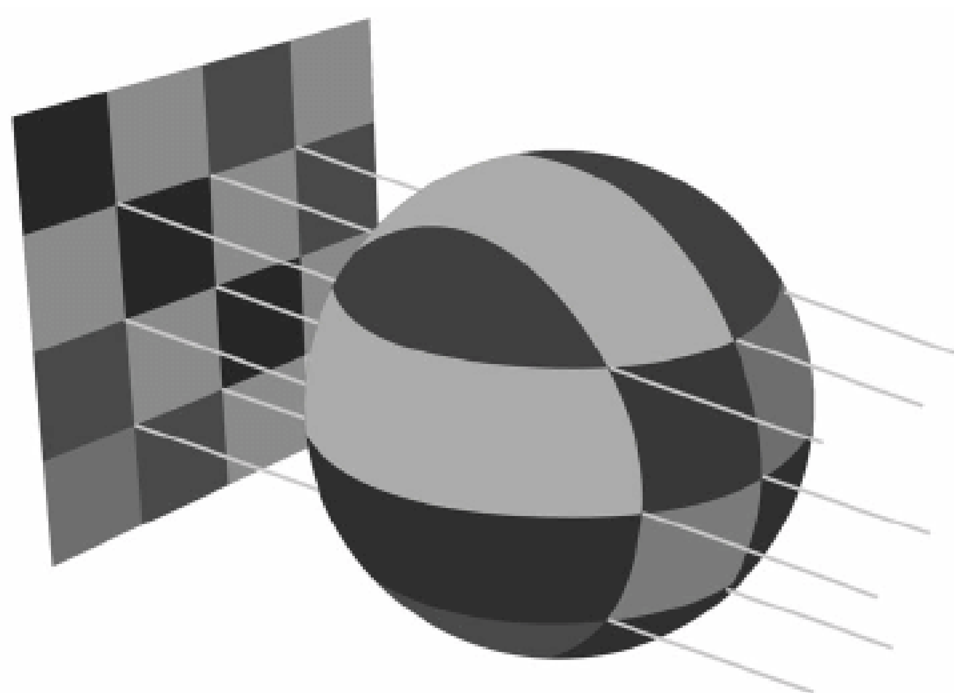


图 20.6　平面贴图

图 20.7 显示了圆柱体贴图，并为当前对象选取了一个轴向。随后，可计算沿该轴向各顶点的距离，该行为将顶点按比例缩放至 t 坐标。对于 s 坐标，则可使用顶点与轴线之间的角度值。

针对球体贴图，可使用经纬度作为数据点与球体间的投影结果。当采用这一方案时，距中心位置间的距离将被丢弃，如图 20.8 所示。

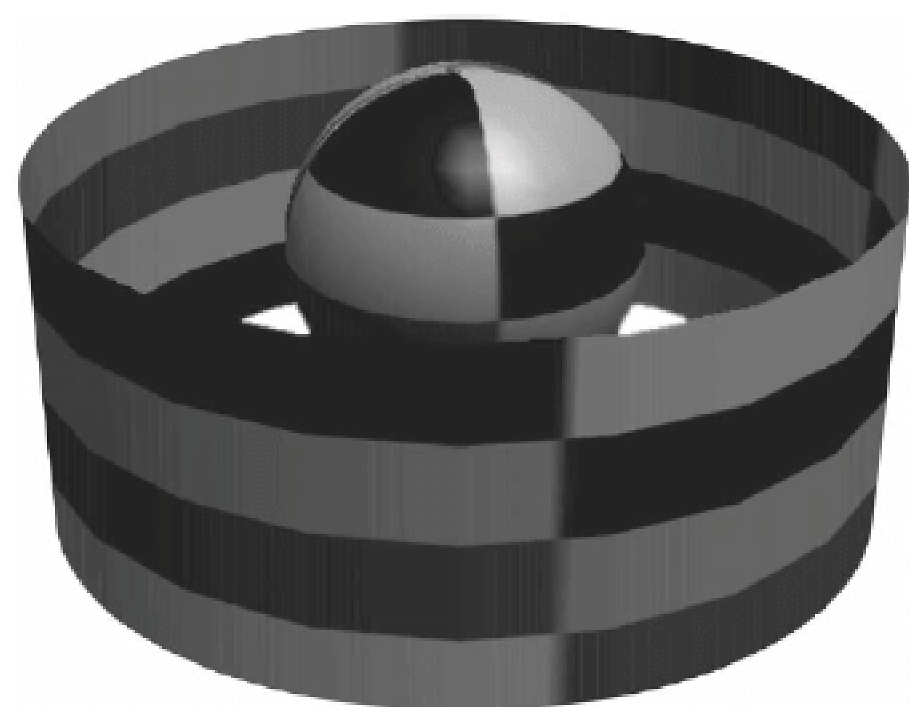


图 20.7　圆柱体贴图

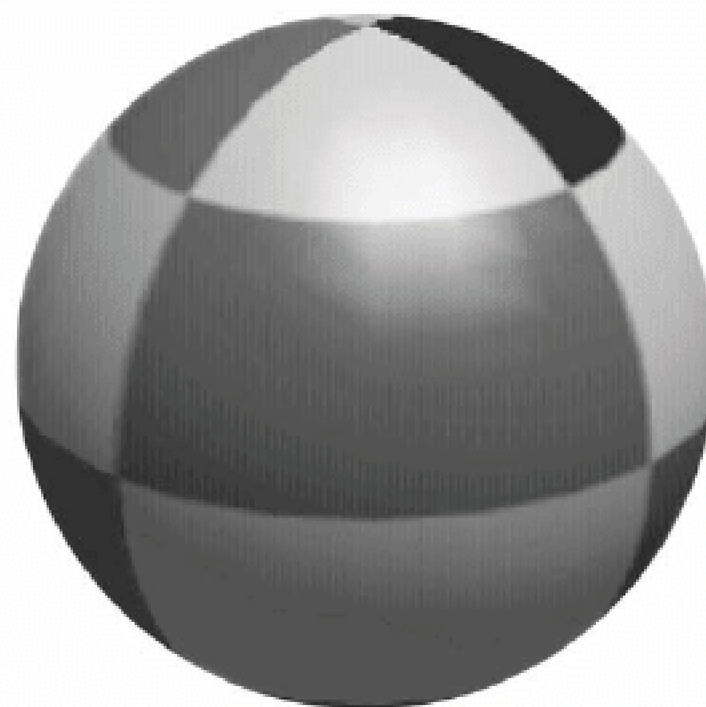


图 20.8　球体贴图

在上述全部示例中，顶点位置直接与纹理坐标关联，需要注意的是，对应位置具有不同的数据值，例如对象的局部几何形状，或者世界坐标位置，因而将生成不同的效果。若采用局部几何坐标，则无论模型如何转换，纹理均保持不变。也就是说，当对模型执行旋转或缩放等操作时，纹理外观将无任何变化。当采用世界坐标时，纹理变化则与对象的放置方式有关。特别地，若围绕某一轴向旋转对象，则纹理保持不变，进而可模拟反射表面效果。当采用纹理表现反射时，其结果通常会朝向同一方向。

反射贴图通常不用于反射效果中，相反，光照图可按相同方式作用于对象上，进而可实现固定阴影效果。读者甚至可尝试将其应用于某些奇特对象上，例如凹凸贴图，其视觉效果通常也较为奇特，例如移动于表面之下的块状物体，这一效果类似于电影《木乃伊》（拍摄于 1999 年）中的食肉甲虫。

20.3.4　纹理链

当使用图像贴图时，贴图与相机之间的距离将会导致问题的出现。若图像距离相机较远，则当前所用表面信息超出了实际需求。若屏幕上的各像素覆盖 100 个不同的纹素，则无须了解各纹素的颜色值。实际上，冗余信息往往会带来负面影响。相反，若图像距离相机过近，则单一纹素将覆盖大量的屏幕空间，这将导致纹素间产生锯齿或锯齿边。

对此，应首先处理锯齿问题。相对于相机，当与近距离对象协同工作时，作为固定颜色平面，通常无须绘制各个纹素。相反，可在纹素间执行平滑插值计算，即双线性过滤机制，如图 20.9 所示。针对图中各像素，需要通过 4 个最近纹素确定一个特定屏幕像素，并在该点处生成全部颜色值得加权平均值。在图 20.9 中，两个立方体数据表面包含相同的 4×4 像素纹理，而左侧纹理设置为双线性过滤。

通过观察可知，图 20.9 所示的双线性过滤并非完美无缺，该方案常使得纹理处于模糊状态。一类替代方案称作过采样，该方案并未在某一像素下计算单一纹素点，并于随后与附近纹素混合；相反，该方案在多个邻近像素间计算纹素点，并执行混合操作。最终结果可描述为，近相机处难

以察觉明显的模糊现象，但却包含清晰的颜色轮廓区域，其间存在多条抗锯齿直线。

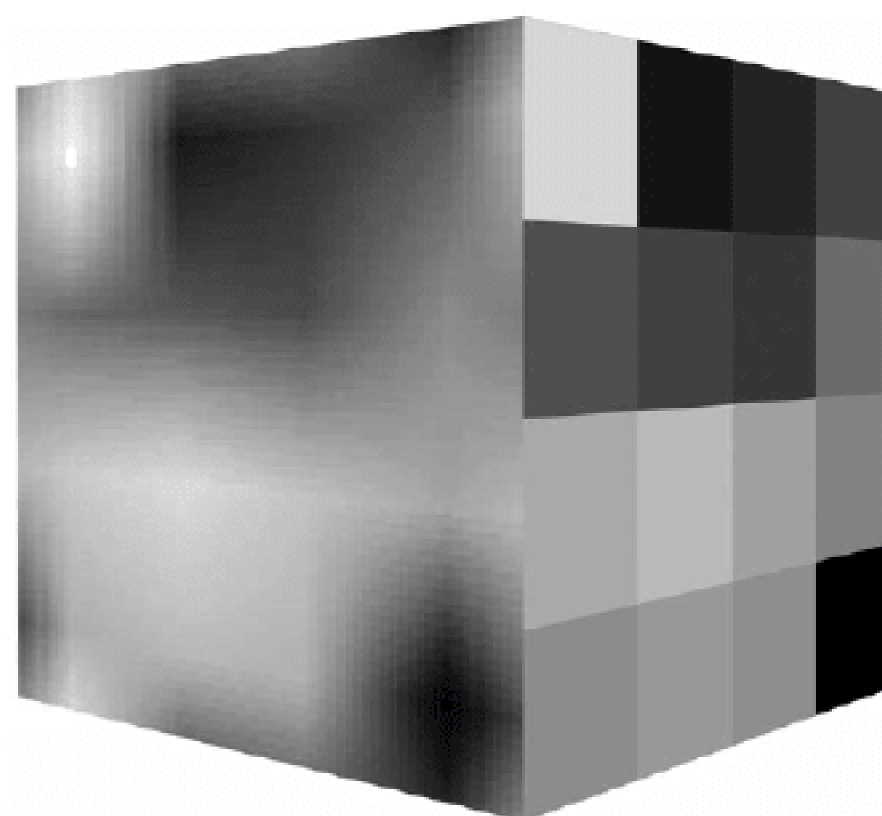


图 20.9　双线性过滤

针对远距离纹理，一类方法是使用纹理链，即表示不同细节级别的、预先计算的一组纹理。当采用纹理链时，假设原始纹理尺寸为 256 × 256，除此之外，还可存储 128×128、64×64 等低分辨率的纹理，直至 1×1 纹素。其中，1×1 纹素等同于全部纹理的颜色平均值。随后，根据多边形所占据的屏幕空间，3D 引擎可选取相应的贴图。相应地，纹理链所占用的内存空间有所增加，但仍在可接受的范围内（不大于 50%）。尽管如此，其处理速度和图像质量均获得了显著的提升。

【提示】术语“纹理链”源自拉丁语，意即多个狭小空间。

当纹理链处理完毕后，依然存在若干问题需要解决，例如不同贴图之间的过渡点。当观察较大平面时，其最近边将通过高质量的纹理贴图予以查看，较远边则使用低质量的纹理贴图。期间，引擎将在不同贴图之间进行切换。当图像质量突然变化时，将产生明显的闪烁现象。为了避免这一问题，可通过三线性过滤机制在贴图间执行插值计算。

三线性插值将在边界处对不同分辨率的数据进行合成计算。在图 20.10 中，可根据邻近两个纹理链计算颜色值，并于随后使用加权平均值确定适宜的颜色，进而实现平滑的过渡行为。

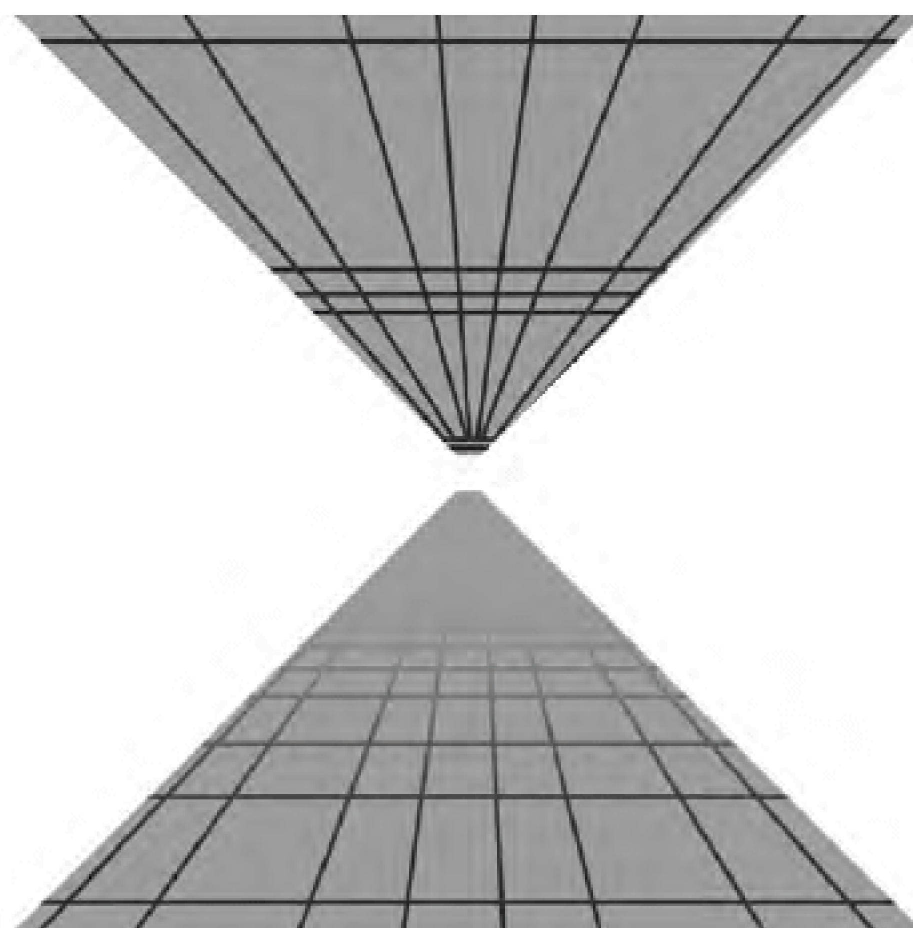


图 20.10　使用/未使用三线性过滤机制时的纹理链效果

某些图形卡支持双线性过滤机制，同时，该机制还可在过渡点附近处选择性地应用三线性过滤。另外，某些图形卡采用各向异性过滤，并合成与视角相关的像素区域，而非整合正方形像素组合。据此，若观察者视线倾斜于对象表面，则该机制通过狭长区域生成合成后的像素颜色。尽管这对处理器提出了较高的要求，但该方案可获得更为真实的视觉效果。

20.4 着色机制

关于模型的子多边形的构建方式，另一种方法是使用着色机制。着色机制根据周围表面并对其颜色执行插值计算，进而得到平滑的对象外观。

20.4.1 Gouraud 和 Phong 着色

Gouraud 着色可视为最为简单的着色方法，该方法根据三角形的 3 个顶点计算正确的颜色值，并于随后在三角形间对其执行插值计算。需要注意的是，Gouraud 着色仅影响材质的固定数据项，且不会受到纹理贴图的影响。

插值计算可通过重心坐标予以实现，该坐标形式类似于齐次坐标。在图 20.11 的左图中，三角形中点 P 的重心坐标(w_1, w_2, w_3)可定义为一个权值集，对应权值置于三角形各顶点上，进而生成点 P 处的重心位置，并可于此处使得三角形处于平衡状态。

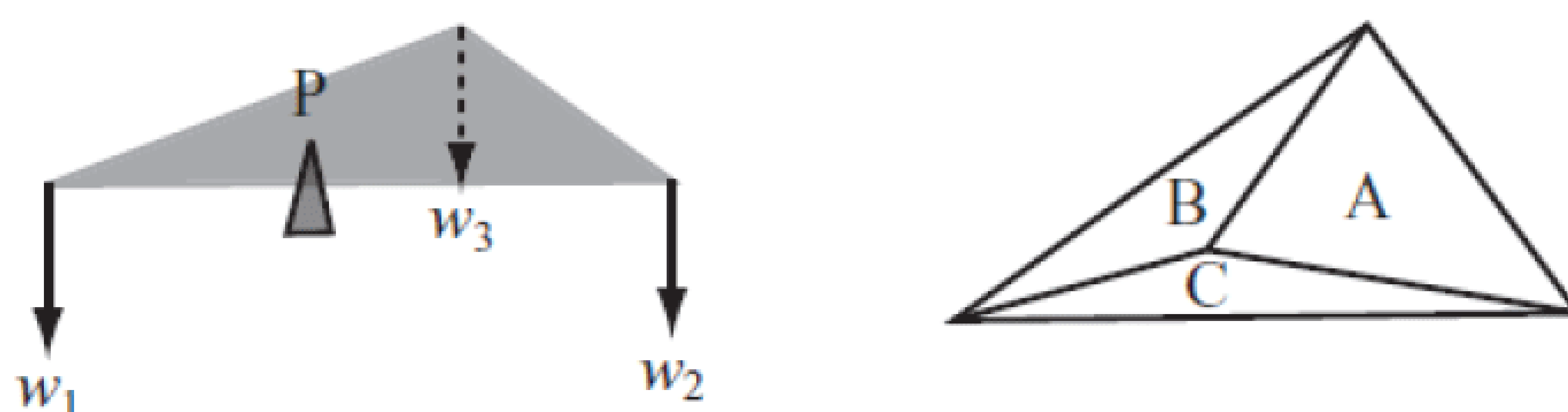


图 20.11　重心坐标

在图 20.11 的右图中，还可根据 A，B，C 区域定义坐标，且令 $w_1 = A$，$w_2 = B$，$w_3 = C$。类似于齐次坐标，鉴于重心坐标不受缩放操作的影响，因而仅存在一种解决方案。在第 17 章曾讨论到，三角形面积等于两边叉积值的一半，因而可通过某一简单函数计算一点的重心坐标，barycentric()函数封装了该方案的逻辑和数学运算，如下所示：

```
function barycentric(p, v1, v2, v3)
  set t1 to v1-p
  set t2 to v2-p
  set t3 to v3-p
  set a1 to t1[1]*t2[2]-t1[2]*t2[1]
  set a2 to t2[1]*t3[2]-t2[2]*t3[1]
  set a3 to t3[1]*t1[2]-t3[2]*t1[1]
  return norm(vector(a1, a2, a3))
end
```

因此，可通过重心坐标（缩放至单位长度）对颜色值进行插值计算。针对三角形各点，可将各顶点的颜色值乘以某一权值并执行加法运算，对应的 colorAtPoint()函数如下所示：

```
function colorAtPoint(pos, vertex1, vertex2,vertex3, color1, color2, color3)
  set coords to barycentric(pos, vertex1, vertex2, vertex3)
```

```
  return color1*coords[1] + color2*coords[2] + color3*coords[3]
end function
```

在图形应用程序中，当采用优化操作并支持整数计算时，上述处理过程将变得更为高效。另外，若已知一点的重心坐标，且全部 3 个坐标均位于 0～1 之间，则该点位于三角形内部。

对于计算能力较强的图形卡，可通过某些额外工作生成三角形的全局凹凸贴图。这里，可对三角形法线执行插值计算（而非计算颜色值并执行插值计算），并以此执行逐像素的光照计算，即 Phong 着色机制。该方案对处理器提出了较高的计算要求，为了节省计算时间，3D 引擎通常采用逐顶点方式计算光照强度，当计算各光源的分布结果时，进一步在三角形中执行插值计算。

20.4.2　顶点法线

上述着色方案取决于各顶点处平滑表面的法线计算，因而会产生处理载荷问题。为了处理这一问题，可通过两种方式计算各顶点处的法线。其中，一类最为简单的方法是计算共享某一顶点的全部三角形的法线“均值”，对于规则形状，该方法工作良好。

一类相对高级的方案则根据各三角形的尺寸对法线进行加权计算，并以此凸显主要三角形的影响力。对此，可采用与重心坐标类似的方法执行叉积计算。

如前所述，针对以 $\mathbf{v}_1$，$\mathbf{v}_2$，$\mathbf{v}_3$ 逆时针方式排列的三角形，可计算 $\mathbf{v}_2 - \mathbf{v}_1$ 和 $\mathbf{v}_3 - \mathbf{v}_1$ 的标准化叉积结果。否则，法线长度值将 2 倍于三角形面积值，在执行标准化操作之前对此类向量执行“均值”计算可得到所需的加权和。

【提示】对于非平滑模型，顶点法线往往不止一个，有时甚至可多达 3 个以上。

20.5　本章练习

【练习 20.1】试编写函数以实现对象表面上的圆柱体、球体以及平面纹理贴图，并在 3D 引擎（若存在）中查看运行结果。另外，读者也可尝试计算网格顶点的 *st* 坐标。这里，难点在于纹理自身相交时奇点（singularity）的处理。例如，球体顶端即可视为一类奇点。

20.6　本章小结

本章深入讨论了光照、纹理以及着色机制。读者学习了真实光照与 3D 环境光照间的关联方式，以及表面与光照间的不同作用方式。除此之外，本章还介绍了材质、图像纹理及其表面投影方式。最后，本章还探讨了着色机制，以及如何通过该机制生成平滑形状的外观。第 21 章将继续考察某些 3D 建模技术，包括贴图的表面生成方式以及水波的建模方式。

至此，读者应掌握如下内容：

- 如何通过可见光谱观察对象。
- 对象与光线之间的作用方式。
- 光照在计算机中的建模方式。
- 环境光、漫反射光、有向光源以及衰减光源的含义。
- 漫反射光、镜面光以及自发光作用于表面的含义。
- 如何创建图像贴图。
- 如何投影图像贴图，进而创建纹理表面、阴影以及反射表面。
- 如何通过纹理链、双线性和三线性过滤以及过采样消除锯齿效果。
- 颜色值与三角形间的插值方式，进而生成平滑的表面外观。

第 21 章　建 模 技 术

本章包含如下内容：

- 概述。
- 数学 3D 建模。
- 动画表面。
- 骨骼动画。

21.1　概　　述

本章讨论基于网格级别的复杂对象创建技术。截止到目前为止，读者已了解了对象于不同位置间的移动方式，且并未考察该对象的创建方式。期间，假设对象由图元（例如球体、盒体）或现有多边形网格构成。本章则进一步讨论基本表面及其定义方式，同时将介绍实时动画表面，例如随波和布料，并对动画角色的创建方式予以简要讨论。

相关技术涉及逆向动力学知识，鉴于可用于各种动画工具中，因而该技术应用较为广泛。该技术背后所蕴含的数学知识留予读者自行考察，本章仅对某些高级话题加以讨论，并为读者日后的学习打下坚实的基础。

21.2　数学 3D 建模

本章首先讨论静态建模，即根据简单组件创建具有真实外观的表面。

21.2.1　旋转表面

首先，可通过车床技术生成旋转表面。当生成旋转表面时，可于开始阶段定义一个单变量函数，该函数在某一特定区间不存在任何根值。随后，可将表面定义为 3D 点集，在某一 x 值范围内以及特定 x 值处，与 x 轴之间的距离可表示为 $f(x)$。如图 21.1 所示，对于三次函数，结果表面为花瓶状对象。其中，函数图位于轴向两侧，若围绕当前轴线旋转函数图，则可得到 3D 形状，即旋转面。

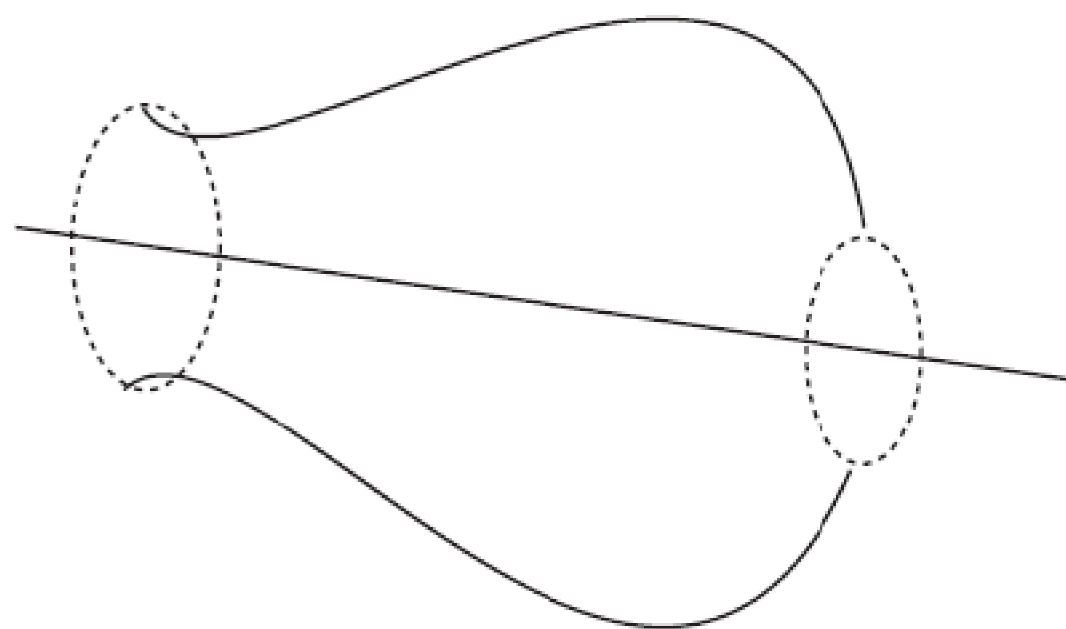


图 21.1　创建花瓶旋转表面

大量的可用形状可生成旋转表面，且沿某一轴向呈完全对称状态，这也意味着，相关对象可在制陶器转轮进行建模。该方案的最大优点是物理元素易于处理，围绕对称轴的旋转表面的转动惯量正比于旋转函数积分 $\int f(x)^2 dx$ 。

类似地，基于旋转表面的碰撞检测也可得到适当的简化。对此，可将表面视为连续的圆锥体截体，且仅关注沿主表面的碰撞（不包括上方和下方）——计算复杂度大多来源于此。

21.2.2　3D 样条

为了创建更为复杂的表面，可将“样条”这一概念引入至三维空间内，进而生成样条表面。在 3D 空间内，B 样条或 NURBS（非均匀有理 B 样条）包含了较为复杂的数学内容。然而，借鉴于前述叙述方式，可于先期考察 Bezier 或 Catmull-Rom 样条，并将其引入至 3D 环境中。

在空间中构建独立曲线可视为最为简单的应用，其复杂之处在于，可在各控制点处定义一条法线。如图 21.2 所示，最终结果类似于 3D 路径曲线，并在空间中扭曲翻转。这里，主曲线基于标准的 Catmull-Rom 样条，使得各控制点为一个 3D 向量，并于随后在各点处生成一个法线向量（与当前曲线垂直并与垂直方向呈正确的角度）。最终结果表示为 3D 环境中的平滑路径。

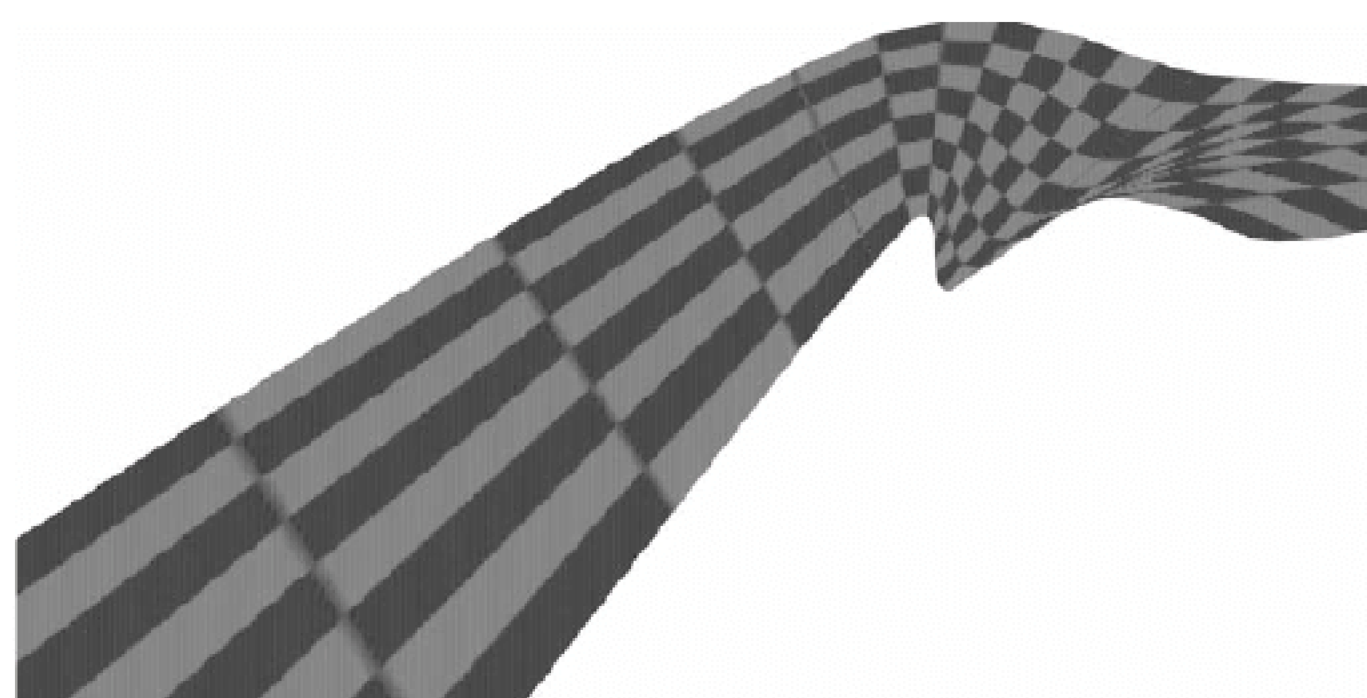


图 21.2　通过样条生成曲线路径

【提示】第 23 章将考察单元拼贴型（tiled）样条，届时将再次分析上述示例。

图 21.2 所示方案的问题在于无法创建完整的表面，最终表面仅涵盖某一特定直线。对此，可构造曲线网格以消除这一问题。最终，可定义 $n \times m$ 个控制点，并通过样条对网格直线插值计算（该网格在某一方向上包含 n 个样条，在另一方向上包含 m 个样条）。针对目标集合，尽管该方案工作良好，但计算量亦不容小视，且两个方向上无法实现真正的整合。

21.2.3 NURBS

虽然 21.2.2 节中所讨论的方案提供了表面处理的良好开端，但在工业标准中，表面往往采用更为高级的处理方案。在 3D 软件包中，NURBS 可视为一类较为常见的表面绘制工具。B 样条封装了 Bezier 和 Catmull-Rom 曲线并兼具多种功能。B 样条可用于表达圆、椭圆、球体以及圆环面（tori）。

【提示】圆环面表示为一类圆环形状，该术语表示为 torus 的复数。

与前述样条曲线不同，B 样条并未采用基于独立三次曲线的逐线段方式加以定义，相反，B 样条采用系列数据值$\{t_0, t_1,\cdots, t_m\}$（针对各 i 值，有 $0\leqslant t_i\leqslant t_{i+1}\leqslant 1$）所定义的独立节点向量予以描述，这将在称作节点的曲线点上构造曲线点集，各点可视为如下结果：将参数 t 设置为某一节点向量值。

由于不同种类的节点向量须通过特定操作行为予以创建，因而节点向量可视为 B 样条的主要方式。当节点以等间隔分布时，该曲线称作均匀曲线，这也是非均匀（NU）NURBS 这一称谓的由来。除了节点之外，曲线的物理形状还可通过多个控制点$\{P_0, P_1,\cdots, P_n\}$创建，且有 $n\leqslant m$。其中，样条阶数等于 $k=m-n-1$。

另外一个所需元素则是函数集，且称作基函数或混合函数，此类函数通过递归方式定义。

首先，针对 $p<j\leqslant k$（且 $\frac{0}{0}$ 定义为 0），有如下方程：

$$N_{i,j}(t)=\frac{t-t_i}{t_{i+j}-t_i}N_{i,j-1}(t)+\frac{t_{i+j+1}-t}{t_{i+j+1}-t_{i+1}}N_{i+1,j-1}(t)$$

随后，针对 $t_i\leqslant t\leqslant t_{i+1}$（否则为 0），则有：

$$N_{i,0}(t)=1$$

【提示】这里可对基向量这一概念稍作回顾，向量空间可扩展至更为抽象的领域，尤其是函数范畴。类似于传统向量，抽象向量空间内的基向量可定义为一个数据元素集，且该空间内的任一元素均可记为基元素倍数的线性和。最终，针对多项式函数空间，基元素可表示为函数 1，x，x^2，x^3 等，且该空间包含无穷多个维度。

上述函数难以实现视觉化描述，下面将引入一个简单的示例。这里，假设 0，0.2，0.5，0.8，1 处包含 5 个节点，且曲线阶数为 2（包含两个控制点），因而需要计算 $N_{1,2}$。根据递归定义可得到如下算式：

$$N_{1,2}(t)=\frac{t-t_1}{t_3-t_1}N_{1,1}(t)+\frac{t_4-t}{t_4-t_2}N_{2,1}(t)$$

再次根据递归定义，可得到下列算式：

$$N_{1,2}(t)=\frac{t-t_1}{t_3-t_1}\left\langle\frac{t-t_1}{t_3-t_1}N_{1,0}(t)+\frac{t_3-t}{t_3-t_2}N_{2,0}(t)\right\rangle$$

$$+\frac{t_4-t}{t_4-t_2}\left\langle\frac{t-t_2}{t_3-t_2}N_{2,0}(t)+\frac{t_4-t}{t_4-t_3}N_{3,0}(t)\right\rangle$$

当前，全部函数降至 $j=0$，这也意味着，可通过基本定义并采用下列不同的计算行为：

- 若 $t<0.2$，则全部函数的求值为 0，因而有 $N_{1,2}(t)=0$。
- 若 $0.2\leqslant t<0.5$，则可得到 $N_{1,0}(t)=1$，因而有：

$$N_{1,2}(t)=\frac{(t-t_1)^2}{(t_3-t_1)(t_2-t_1)}=\left(\frac{(t-0.2)^2}{0.6\times 0.3}\right)$$

- 若 $0.5\leqslant t<0.8$，则可得到 $N_{2,0}(t)=1$，因而有：

$$N_{1,2}(t)=\frac{(t-t_1)(t_3-t)}{(t_3-t_1)(t_3-t_2)}+\frac{(t_4-t)(t-t_2)}{(t_4-t_2)(t_3-t_2)}$$

- 若 $0.8\leqslant t<1$，则可得到 $N_{3,0}(t)=1$，因而有：

$$N_{1,2}(t)=\frac{(t_4-t)^2}{(t_4-t_2)(t_4-t_3)}$$

需要注意的是，若 $j=k$，则基函数包含函数 $N_{i,0}$ 的 $k+1$ 引用。另外，函数完全独立于控制点的数据值，这也是节点向量对曲线产生深刻影响的原因所在。进一步讲，虽然基函数定义具有一定的计算复杂度，实际上，其实现过程较为简单。NURBSbasisFunction()函数封装了递归计算，如下所示：

```
function NURBSbasisFunction( i, j, t, knotvector)
  if j=0 then //bottom out recursion
    if t<knotvector[i+1] then return 0
    if t>=knotvector[i+2] then return 0
    return 1
  end if
  //otherwise recurse
  if (knotvector[i+j+1]-knotvector[i+1])=0 then set a to 0
      otherwise set a to (t-knotvector[i+1]) /
                    (knotvector[i+j+1]-knotvector[i+1])
  if (knotvector[i+j+2]-knotvector[i+2])=0 then set b to 0
       otherwise set b to (knotvector[i+j+2]-t)/(knotvector[i+j+2]-knotvector
                          [i+2])
  return a*NURBSbasisFunction(i,j-1,t,knotvector)+ b*NURBSbasisFunction(i+1,j-1,
          t,knotvector)
end function
```

【提示】回忆一下，本书所定义的数组从索引 1 处开始。

待基函数定义完毕后，则可得到 B 样条曲线方程，如下所示：

$$C(t)=\sum_{i=0}^{n}P_iN_{i,k}(t)$$

尽管上述函数易于计算，但与前述样条相比，其操作过程较为抽象。即使如此，读者仍可尝试对其施加局部操作行为。其中，各曲线线段因受到附近控制点的影响，各基函数（以及各控制

点）仅对 k+1 曲线段有所贡献。

这里的问题是，如何处理 NURBS 中的有理 B 样条（RBS）部分？若采用齐次坐标定义控制点，则可将坐标视为一个 3D 向量以及一个标量值 w_i，即控制点的权值。对此，可在全部 B 样条上针对各控制点将 w_i 设置为 1 得以实现。调整 w 值则可获得一定的控制权，此处，w 为 1 的曲线称作非有理曲线，而 w 可变化的曲线则称作有理曲线。因此，NURBS 曲线的通用形式如下所示：

$$C(t)=\frac{\sum_{i=0}^{n} P_i w_i N_{i,k}(t)}{\sum_{i=0}^{n} w_i N_{i,k}(t)}$$

【提示】NURBS 为单数名词，然而，该术语也常用作形容词，例如 NURBS 表面或 NURBS 曲线。

相对于 NURBS，有理样条的优势在于全转换的不变性。若对空间执行转换，则位于 NURBS 表面某一侧的对象在转换操作完毕后未必会位于同一侧，而有理 B 样条则不会出现这种情况，即对象在转换后位于同一侧。然而，NURBS 的不变性体现于仿射转换和全转换。

与其他样条相比，NURBS 的优点在于转换特征易应用于表面上（而非直线），这一联系方式可通过添加第二个求和项实现，如下所示：

$$S(s,t)=\frac{\sum_{i=0}^{p}\sum_{j=0}^{q} P_{i,j} w_{i,j} N_{i,k}(s) N_{j,l}(t)}{\sum_{i=0}^{p}\sum_{j=0}^{q} w_{i,j} N_{i,k}(s) N_{j,l}(t)}$$

此处，控制点以网格方式排列，并在两个方向上分别包含 p 和 q 个控制点，以及两个阶数分别为 k 和 l 的节点向量。在练习 21.1 中，读者可尝试通过显式算法转化这一描述。

21.2.4 基于正弦和余弦函数的表面

针对通用实体对象的创建，NURBS 表面可视为一类较好的工具。当然，某些场合下也存在相应的替代方案，例如构造无限地表平面或高度图时。3D 环境中的高度图定义为二变量函数，针对各 x 和 z 值，该函数将生成单一 y 值。由于可据此确定全部地表表面（仅通过少量信息），因而当通过简单方法产生此类函数时，该方案十分有效。

第 15 章曾讨论了一种高度图生成方法，该方法使用了三角函数组合。当多个正弦波叠加时，最终结果将生成一类较为复杂的模式。如图 21.3 所示，若在两个方向上进行组合，则可得到包含随机外观的山地地形。

当构建随机地形时，三角函数组合可保证山脉或山谷等地形的最大和最小高度值。若叠加多个波形，则对于山脉地形而言，最大可能高度可表示为振幅之和。除此之外，该方案无须存储全部地形的高度图，对应函数于先期定义完毕，因而可在后期绘制较远处的区域，且无须对其进行显式存储。

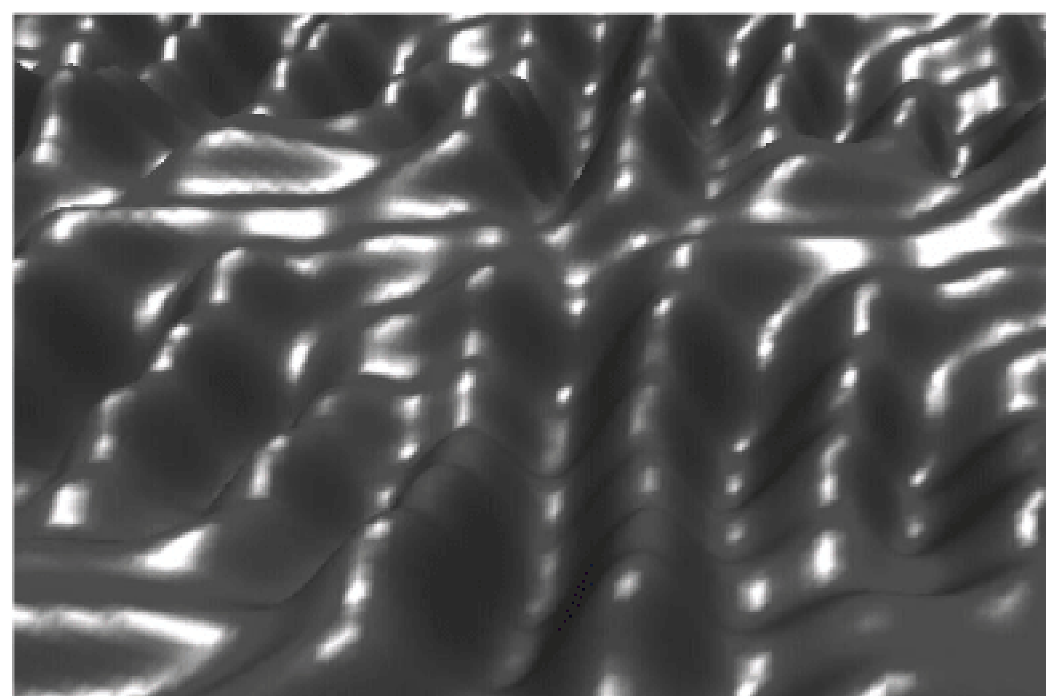


图 21.3　使用正弦和余弦函数生成的地形

21.2.5　细分操作

当采用算法确定表面时，无须通过无限多个点描述平滑、波状表面。在实际操作过程中，3D 引擎并未采用精致的方法显示某一表面，相反，表面被转化为一组多边形集合。如果将表面存储为精确的平滑曲线，则可在运行期内生成多边形网格，即细分过程。

当对表面进行细分时，可于先期计算网格上多个数据点的坐标。对于 NURBS 表面，通常情况下，数据点在 s 和 t 值方向上等间隔排列；或者，对于地表表面，数据点在 x 和 z 值方向上间隔排列，例如前述内容所讨论的三角形表面。随后，此类数据值连接为三角形，进而转化为网格。取决于所使用的 3D API，对应传递方式可表示为：独立三角形或顶点集，以及一组定义了顶点与三角形的连接方式的数值集。例如，DirectX 将三角形定义为 3 个顶点构成的列表、三角带（通过单一点扩展三角带）或者源自某一点的三角扇。

这里的问题是，既然将表面转换为网格，为何不存储一个网格并对其进行处理？其原因主要可描述为：类似于矢量图和位图之间的差异，转换至网格通常可占据较少的内存空间。此处，可将某一形状存储为一种描述方式，而非预定义的点集，当几何形状较为简单时，该方案十分有效。若几何形状相对简单，则 NURBS 则显得大材小用，因而可采用一类简单的描述方案。例如，一类简单的操作可描述为：半径为 2 的球体。另外，如果形状复杂（例如游戏角色），或者形状本身由多个多边形构成（例如钻石），则多边形-多边形描述将更为适宜。在某个中间阶段，尽管 NURBS 或类似系统有可能节省内存空间，但考虑到引擎通常根据相关公式建模，因而这将增加引擎的数据加载时间。

网格的另一个优势则是缩放操作，进而可对网格对象执行细化操作。相应地，网格的细节内容常称作细节级别（LOD），即网格构造过程中所使用的多边形数量。如果终端用户使用运行速度较快的机器，则可生成具有丰富细节内容的网格；否则，终端用户需要构造相对简单的网格。无论如何，多边形都需要基本曲线进行计算，这也意味着，最终结果仅为真实表面的近似外观。另外，还可根据对象与相机之间的距离动态调整网格的 LOD，并采用纹理链对其进行处理。实际上，LOD 常用于描述不同的纹理链级别。

大多数引擎可自动计算 LOD 网格，并可根据距离实现网格间的切换操作。有时，模型距离变化可将网格调整至简单形式。

缩放操作的优势在于，针对自动生成的表面，该方案可计算其上特定点处的法线，并可通过偏导数予以实现（读者可参考第 6 章以获取与偏导数相关的内容），进而可得到与表面相切的两

个向量，如图 21.4 所示。随后，可执行叉积计算获得该点处的法线。

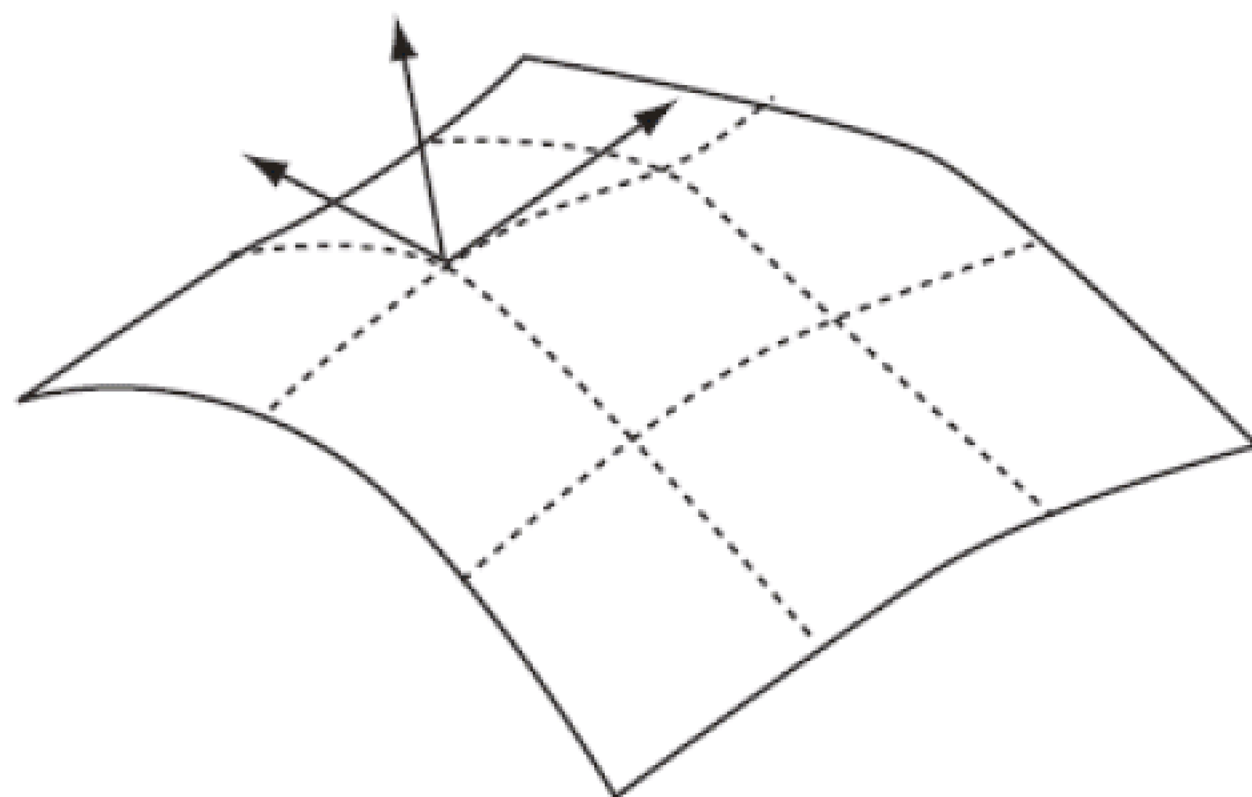


图 21.4　计算表面法线

在 NURBS 表面中，各方向上基函数的偏导数可通过阶数为 $k-1$（或 $l-1$）的独立多项式加以描述，因而各节点跨度保持一致。随后，可将多个偏导数整合至基于整体表面函数的偏导数中。最终结果可描述为，对于各网格顶点，除了计算其位置数据之外，还需进一步计算其法线。类似地，还可对三角表面计算相应的法线数据。

类似于 Bezier 曲线，上述表面的优势也体现于碰撞检测过程中。然而，若采用多边形-多边形碰撞检测方案，则计算复杂度将显著增加。

21.3　动画表面

鉴于实时构造过程并不复杂，因而可考虑在各时刻间生成复杂表面，进而使得表面可在一段时间内发生变化。该问题涉及多个话题，下面对此逐一加以讨论。

21.3.1　布料和头发

如果游戏角色衣着宽松，则需要对布料进行适当计算，其计算量不容小视。纵观游戏发展史，此类宽松型布料较少出现于游戏中。然而，随着计算机运算速度的不断提升，宽松型服饰所产生的问题逐渐弱化，甚至可出现于实时计算中。

关于布料、头发以及皮肤的制作，对于开发人员而言可以说是挑战与机遇并存。在先期阶段，基本技巧可用于相关构造过程中，在第 16 章曾讨论到，此类技巧涉及耦合振荡器的创建。如图 21.5 所示，布料可视为通过弹簧彼此连接的粒子网格。对于绑定于弹簧系统上的粒子，前述章节曾对此有所提及。尽管大多数物理 API 可生成虚拟弹簧，但读者依然有必要了解这一类系统的最佳构造方式。

当构造弹簧系统时，一类关键因素则是真实布料间的纤维多采用交错方式加以编织。但在不同方向上拉伸布料时，交错编织方式使得纤维具有不同的运动行为。当作用力沿纤维的经线或纬线方向作用时，布料较少出现整体拉伸这一现象。若沿斜线拉伸布料，即对角线方向，则通常易

于操作。此时布料类似于篱笆格子，如图 21.6 所示。

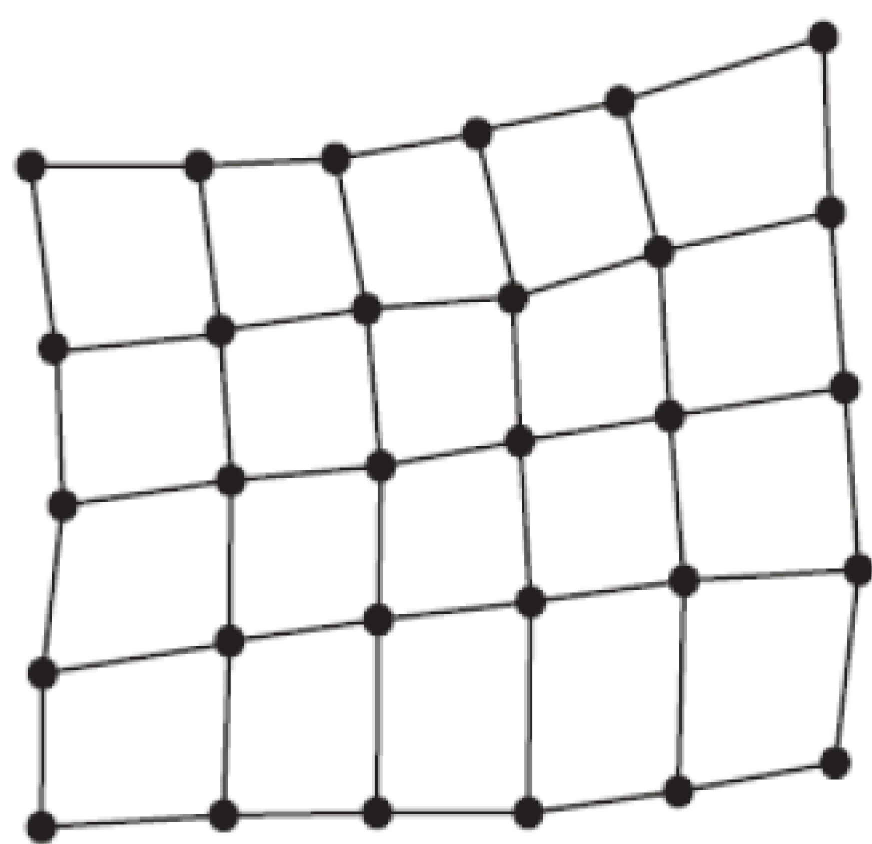


图 21.5　布料模拟

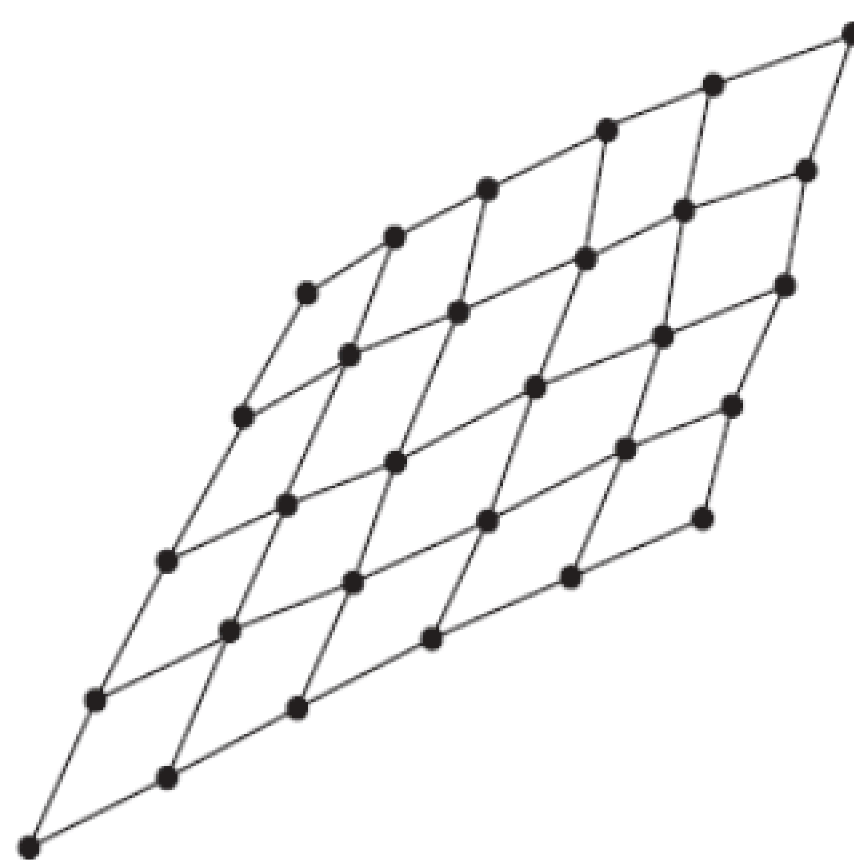


图 21.6　沿对角线方向拉伸布料

布料的动画建模可描述为按网格排列的一组不可伸缩的弹簧，与普通弹簧相比，这一类弹簧则难以模拟。实际上，若可伸缩弹簧的弹性系数趋于无穷大，则对应结果等价于不可伸缩弹簧。在计算机模拟中，构建此类弹簧（特别是耦合网格）将会即刻得到某些反馈问题，此类问题源自误差积累，并导致模拟系统失去控制，即使添加阻尼机制也无济于事。实际上，阻尼机制使得情况变得更加糟糕。

为了克服这一问题，可将布料视为橡胶材质，且橡胶沿各方向均等伸缩，如图 21.7 所示。其中，弹簧以网格方式排列，并设置为较高的弹性系数。除此之外，还可向网格系统中加入支撑条。当系统处于平展状态时，弹簧的自然长度等于其自身长度。

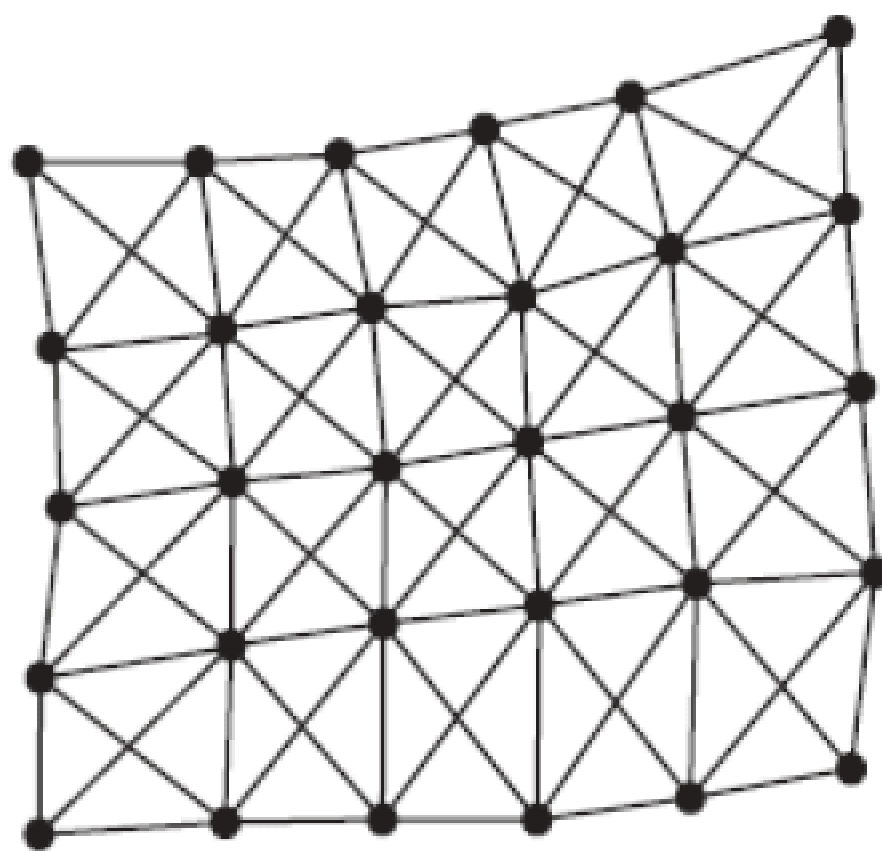


图 21.7　包含支撑条的橡胶材质

为了使模拟过程更加接近于皮肤而非布料，可添加额外的弹簧组，并与基本表面的各顶点进行绑定，此类 0 长度弹簧有时也称作阻尼缓冲器。若表面未位于最佳介质中，则可通过链式方式连接弹簧，该方案常用于模拟头发或部分绳索。

当采用橡胶材质表面时，则可构造悬挂、摆动、褶皱等真实效果。针对褶皱效果，须包含与顶点处粒子的碰撞检测。除此之外，唯一复杂之处在于用户交互计算。由于鼠标操作在模拟环境中并不存在任何物理限制条件，因而用户可轻易地生成并非真实存在的模拟环境。例如，可大范围地拖曳弹簧上的粒子，并使其超出自身的弹性极限。当然，此类问题易于解决。在某些场合下，若用户仅拖曳部分表面，而非直接拖曳粒子，则作用力不可超出设置于粒子上的最大值，并在鼠标指针所设定的方向上进行操作——这将有效地限制表面相对于平衡位置的运动量。

21.3.2 水波

在计算机图形学中，水波与布料之间并无明显区别。一类具有真实感的水波表面将使用到耦合振荡器。针对水面附近点的耦合振荡器，其作用力计算十分复杂，并涉及到重力、压力以及表面张力的合成计算。

一类替代方案则是直接对表面波形进行建模。类似于前述三角形表面，波形表面可通过一系列的波函数加以定义，且函数间可单独执行计算。其中，特定点处的表面高度可视为该点处的波形和。

当采用波函数或振荡机制时，建模的准确性取决于具体方案。对于某些简单方案，可将表面顶点设置为基于简谐振动（SHM）的独立振荡器，且不同顶点具有相同的振荡频率，但振幅和相位彼此不同。其中，变化可通过函数或纹理予以确定，这也是游戏中较为常见的运动波形表面的模拟形式。尽管易于使用且具有较快的计算速度，但这一类系统往往缺乏应有的灵活性，当与用户交互时尤其如此。同时，此类系统不会受到模拟环境中其他对象的影响。

当制作具有真实感的波形表面时，可对其进行直接建模，对应波形可包含不同频率，并在一段时间内发生变化，这也可视为该方案的一个优点。例如，为了有效地模拟不同的天气条件，可在一段时间内增加波形的强度。表面上的波形分为两种类型，即以直线方式移动的简单平行波前，以及源自点源的圆形波前（例如石子投入池塘中），两种表面波都可生成与玩家行为对应的波形。

前述内容所讨论的方法仅生成模拟波，也就是说，此类波不包含波峰或浪花。当水波从深水区移至浅水区时，则会产生波峰和浪花。由于波形不会降至最低负振幅，因而不再以对称方式运动。因此，能量将转换至波形的顶端处，此时，波形前移而非上下运动。上述行为的建模涉及大量的计算，游戏开发人员指出，实时游戏无法提供此类所需的处理能力。然而，在流体动力学领域，此类计算却又不可或缺。类似地，此类计算也出现于电影工业中，且渲染操作无法通过实时行为予以实现。

【提示】*在计算机图形学中，另一类计算开销较大的视觉效果则是水面的反射和折射。尽管存在多种处理方案，但其详细内容则超出了本书的讨论范围。*

21.4 骨骼动画

本章最后一项内容是骨骼动画，在骨骼建模工具的支持下，尽管动画制作者无须了解骨骼创建过程中的细节内容，但理解其数学和编程内容依然十分重要。

21.4.1 与骨骼协同工作

作为角色动画的一种处理方法，骨骼系统历时已久。据此，角色可通过一系列表达骨骼的直

线加以定义。如图 21.8 所示，对应直线以父子关系排列。为了创建正确的模型，骨骼须实现具体化外观，且对应模型可在一段时间内产生变化，相应的调整步骤涉及骨骼的旋转操作。其中，各骨骼相对于父节点进行旋转。

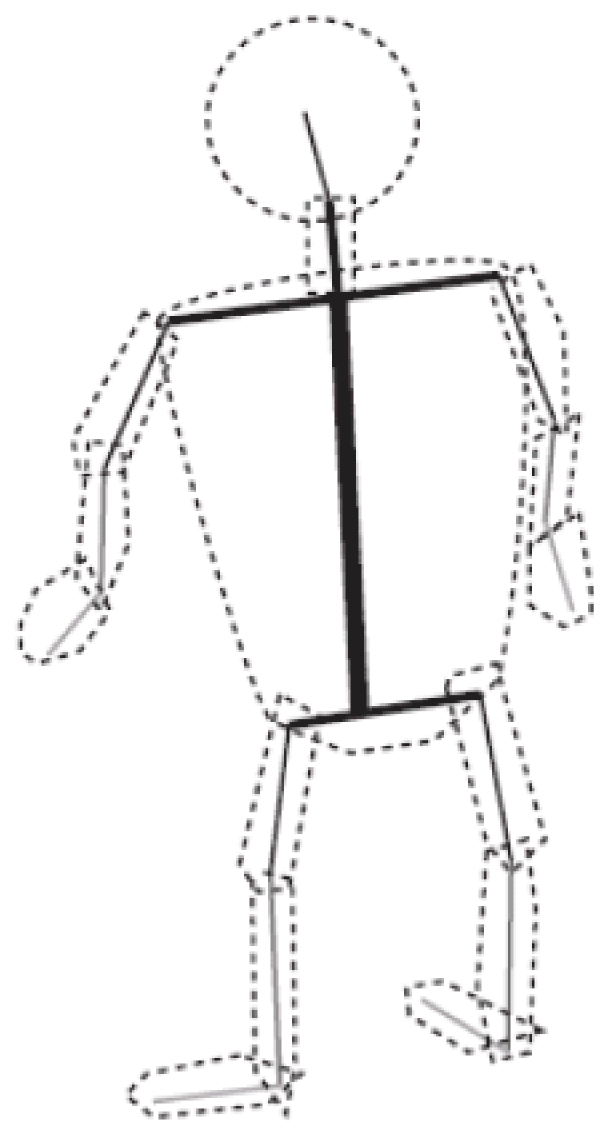


图 21.8　骨骼系统

由于对象表面需以建模实现，并在骨骼弯曲时须保证关节处的皮肤不会产生变形。基于骨骼的建模方式相对复杂，大多数建模软件可对此进行自动处理。当前，读者仅需了解骨骼的工作方式即可。

在开始阶段，可通过某些基本方案创建动画，例如预置动画序列，其中包括跑、跳跃以及跌倒等动作，并通过运动捕捉系统以及真实角色予以记录。另外，还可对各骨骼以实时方式执行动画操作。当采用预置方案时，对应操作过程易于实现且应用广泛。当采用直接动画方案时，其过程相对复杂且常用于布娃娃（ragdoll）动画。对于布娃娃动画，角色肢体在重力作用下以一组连杆方式运动，且不包含任何肌肉动作。

由于模型关节受到不同形式的约束，因而构造具有真实感的布娃娃任务十分困难。例如，人体包含肩膀等一系列球窝式关节，并可在两个方向上自由旋转，而在第三个方向上仅包含有限的运动行为。其他关节，例如膝盖关节，则可视为一类简单的铰状关节，且仅包含一个维度的旋转行为。前臂的旋钮方式则通过两个独立的骨骼予以实现（而非肘关节运动）。这一类约束条件可通过标准的建模软件包以及编程方式实现，但实时工作模式将占用大量的计算开销。

布娃娃角色的运动行为隶属于动力学范畴，后者用于处理系统的运动过程，且无须考察与此相关的物理定律。关于角色动画，动力学处理骨骼的建模方式，以及相对于其他骨骼的运动方式。因此，基于动力学的程序设计较为复杂，期间结合了刚体运动和 3D 碰撞，其数学问题多采用数值方案加以解决。在各时间帧内，可计算对应的动量值和能量值，并以此移动骨骼对象。

作为动力学的展示示例，下面考察一类较为简单的情形，如图 21.9 所示。在 2D 示意图中，当前系统由销钉（枢纽）连接的两块骨骼构成，且线速度和角速度表示为 $\mathbf{u}_i$ 和 ω_i，质量和转动惯量则分别表示为 m_i 和 I_i。针对 3D 动画，还须进一步了解围绕 3 个主轴方向的转动惯量。这里，假设销钉与肢臂中心位置之间的距离为 $\mathbf{x}_i$，全部旋转行为出现于两个维度上且不存在碰撞行为。其中，肢臂于上方或下方彼此经过。

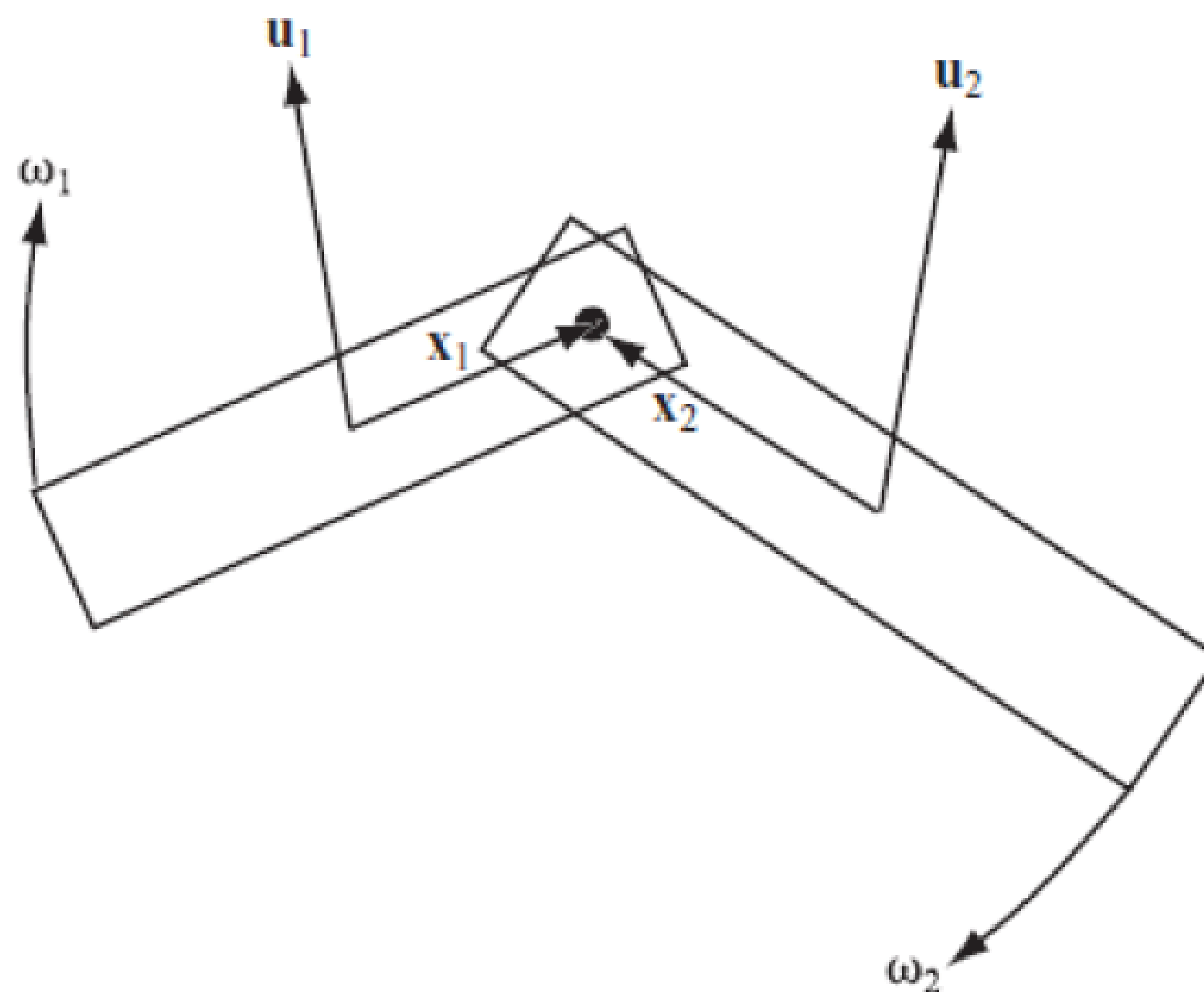


图 21.9　简单的布娃娃示例

这里，可根据 $r_i=|\mathbf{x}_i|$和 $\mathbf{t}_i=\dfrac{(-x_{i2}x_{i1})^{\mathrm{T}}}{r_1}$ 值进行考察，对应值分别计算销钉距中心位置间的距离，以及与该中心相切的顺时针向量。不同肢臂的枢纽机制表明，碰撞点的本地速度彼此相等。对此，可将特定点的本地速度记为肢臂运动函数，如下所示：

$$\mathbf{w}_\mathrm{i} = \mathbf{u}_\mathrm{i} + \mathrm{d}_\mathrm{i}\omega_\mathrm{i}\mathbf{t}_\mathrm{i}$$

在任意时刻以及特定枢纽处，该值针对全部骨骼对均相等。另外，针对新的线速度 $\mathbf{v}_i$ 和角速度 φ_i，可通过下列内容获得相应的方程组。

- 根据能量守恒定律可得到如下方程：

$$m_1|\mathbf{u}_1|^2+I_1\omega_1{}^2+m_2|\mathbf{u}_2|^2+I_2\omega_2{}^2=m_1|\mathbf{v}_1|^2+I_1\varphi_1{}^2+m_2|\mathbf{v}_2|^2I_2\varphi_2{}^2$$

- 根据线动量和角动量（若不存在外部碰撞），可得到如下两个方程：

$$m_1\mathbf{u}_1 + m_2\mathbf{u}_2 = m_1\mathbf{v}_1 + m_2\mathbf{v}_2$$

$$m_1r_1\mathbf{u}_1 \cdot \mathbf{t}_1 + m_2r_2\mathbf{u}_2 \cdot \mathbf{t}_2 = m_1r_1\mathbf{v}_1 \cdot \mathbf{t}_1 + m_2r_2\mathbf{v}_2 \cdot \mathbf{t}_2$$

- 根据枢纽机制，可得到如下方程：

$$\mathbf{v}_1 + d_1\varphi_1\mathbf{t}_1 = \mathbf{v}_2 + d_2\varphi_2\mathbf{t}_2$$

上述方程包含 4 个未知项，其中两项为向量。尽管求解过程较为困难，但依然可行，难点在于半径和切向部分之间不存在清晰的划分。

除此之外，上述方程可能存在数值误差积累现象，并导致骨骼间处于分离状态。对此，一类替代方案利用了骨骼的父子关系，并将某一根骨骼视为根节点，且其他骨骼隶属于该节点之下。因此，全部工作仅需计算各骨骼围绕枢点的相对于父节点的角速度。尽管对应方程相对简单且符合要求，但其构造过程稍显困难，此处并不打算对其予以深究。

21.4.2　逆向动力学

人类的大脑时刻处于工作状态，并对各种动作行为控制身体中的不同骨骼，该情形也适用于

模型的动画行为，这使得当前内容转化为逆向动力学（IK）问题。例如，控制哪些骨骼可准确地拾取一个茶杯？

此类问题多见于机器人学领域，并可应用于动画操作中。相应地，对应求解方案也多种多样。在第 26 章中讨论人工智能（AI）行为时，将对此进行讨论，并展示与此相关的通用算法，当前内容仅考察与逆向动力学相关的某些基本示例。

图 21.10 显示了经适当简化后的骨骼系统，该模型与图 21.9 所示内容相似，即一对经关节连接的骨骼。其中，左侧骨骼于左端固定，而另一端则可自由移动；第二块骨骼可于两端自由运动，并连接至第一块骨骼的自由端。这里，假设需要通过第二块骨骼的端点触及点 P。

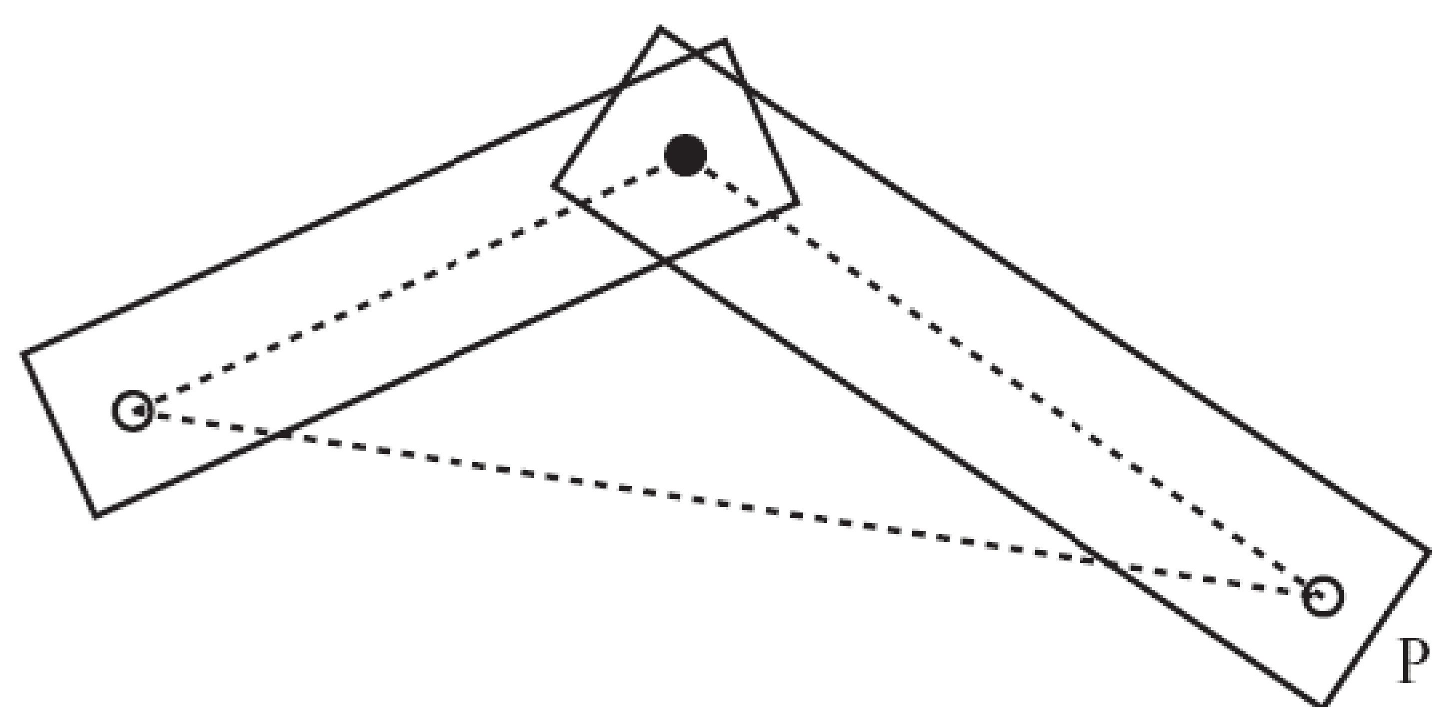


图 21.10　简单的 IK 问题

在图 21.10 中，获取达到点 P 的最终结构相对简单，并涉及余弦定理的应用。根据关节和 3 个端点构成的三角形，其 3 条边的长度均为已知内容。然而，这一简单的问题依然包含了某些复杂之处。除了计算端点之外，还需进一步计算到达目标点的最佳路径，即最小化初始结构和最终结构之间的移动量。图 21.11 显示了一类“冗余”运动方式。

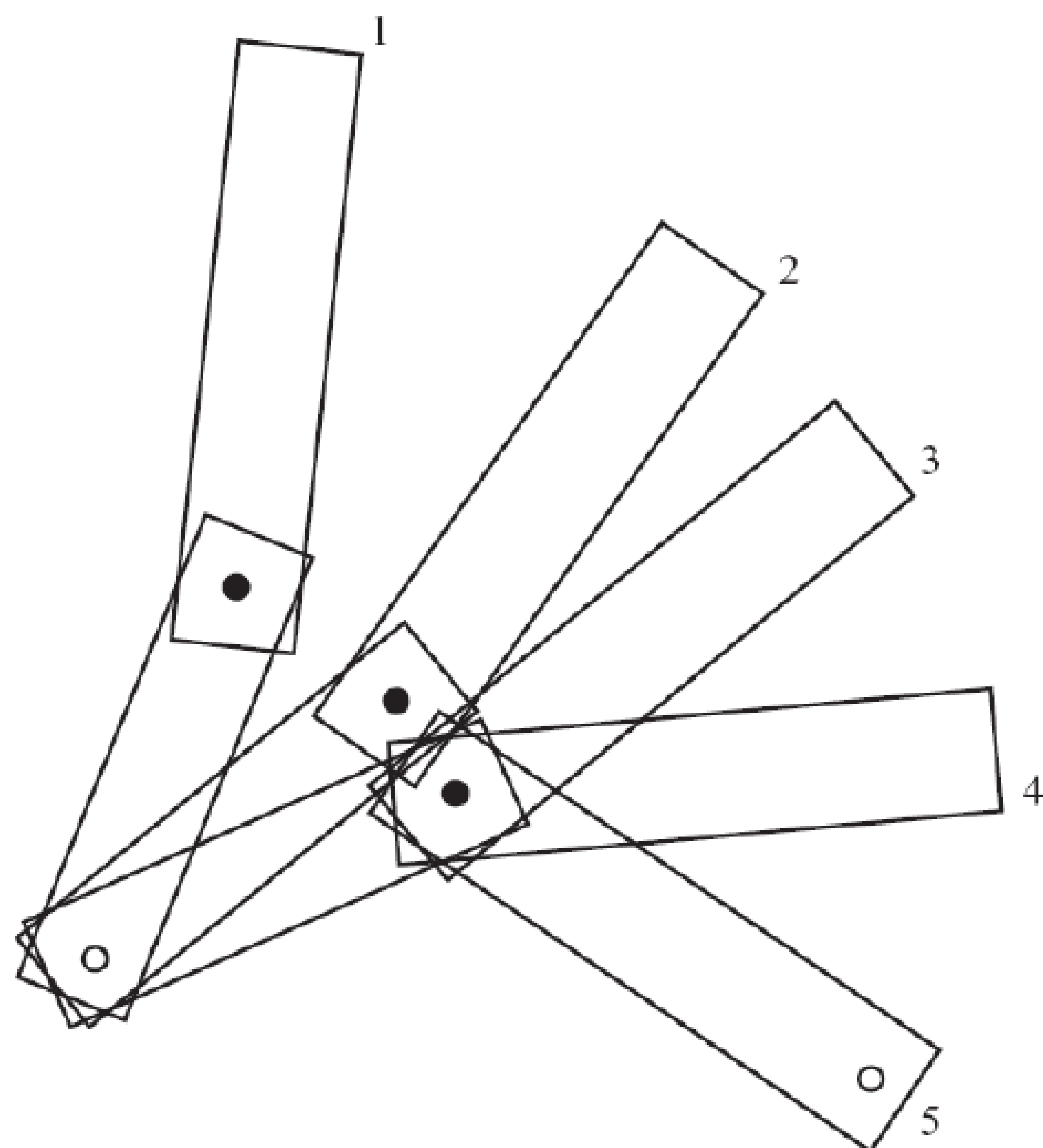


图 21.11　IK 问题的一类简单解决方案

一种消除冗余运动的方法是生成全运动路径，其目的在于使得第二块骨骼的自由端可沿简单直线（即初始点与 P 之间的直线）运动，如图 21.12 所示。该方案的优点在于，相关对象的运动

幅度较小，并以渐进方式抵达目标位置。若端点可达，则该方案针对当前两块骨骼可得到有效解。然而，对于关节有限运动的真实角色，该处理方案可能无法满足物理真实性。

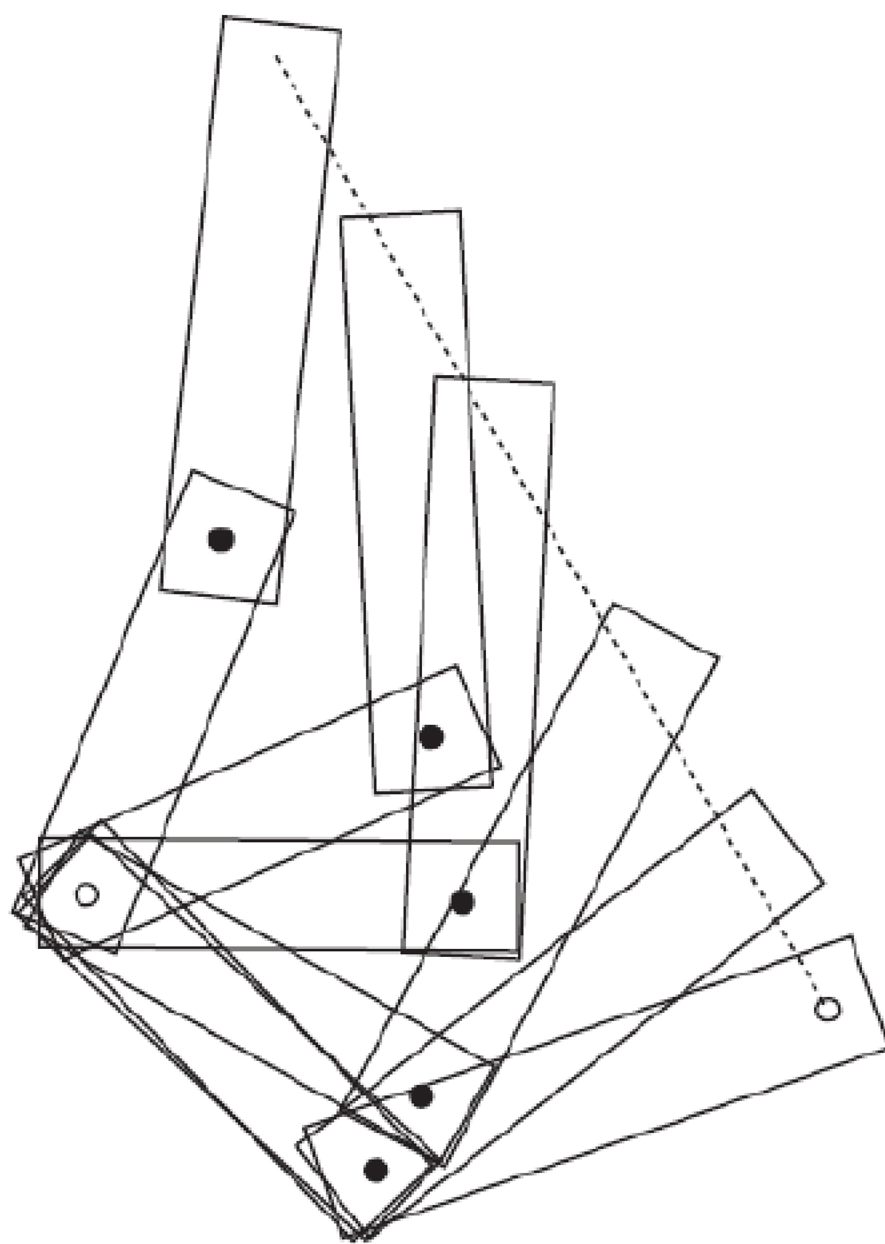


图 21.12　IK 问题的最佳处理方案

此处可能过分强调了逆向运动学的复杂性，实际上，当骨骼数量大于 2 时，通常存在无穷多个可能方案求解 IK 问题，但选择适宜的骨骼运动路径并非易事，其中，骨骼在一次操作中少量移动，并以此方式朝向目标位置。一类简单方法可描述为，根据与目标位置距离之间的比例，在每个时间步内适当地增加骨骼的旋转量，以使其逐步朝向目标位置。读者可尝试练习 21.2 以对该问题进行求解。

上述方案也适用于三维环境，但自由度数量的增加使得获取真实的运动行为变得更加困难。对于非单链或需要实现碰撞躲避的骨骼系统而言，该方案的成功几率将有所降低。

21.5　本 章 练 习

【练习 21.1】尝试编写程序以绘制二维 NURBS，并可适当移动控制点、调整节点向量。另外，有兴趣的读者还可进一步实现三维 NURBS 表面。

【练习 21.2】试编写迭代函数 IKapproach(chain,target)，调整简单 IK 链以使相关对象可到达特定点。该函数可接收某一特定形式的骨骼链，并以小幅度移至目标位置处。据此，经一系列操作后，该函数可调整骨骼链以使其移至目标位置。

21.6　本 章 小 结

关于 3D 技术，本章简要讨论了表面处理方法，其中包括如何通过数学技巧以及各种方法（尤

其是 B 样条）定义复杂表面，以及表面细节内容的调整方式。除此之外，本章还介绍了动画表面以及骨骼系统。第 22 章将借助于游戏环境考察某些算法技术。

至此，读者应掌握如下内容：

- 如何旋转表面、NURBS 以及三角函数定义 3D 对象。
- 如何通过数学描述并以不同细节级别绘制对象。
- 如何对表面实施动画操作以对水波或布料进行模拟。
- 如何构造骨骼系统并以此模拟布娃娃系统。
- 动力学以及二维环境中简单逆向动力学的求解方案。

第 5 部分　游戏算法

截止到目前为止，大多数数学内容均具有一定的通用性，并与真实世界的物理学、几何形状以及计算机模拟方式关系紧密。本书最后一部分内容讨论某些与特定计算和游戏相关的数学理念。由于各项内容均可自成话题，因而本章仅对其进行简要介绍，旨在为日后工作中提供某种参考。

另外，本章还将考察优化技术以及物理计算（特别是碰撞行为）的简化方案。第 23 章和 24 章将分别讨论游戏关卡设计，其中包括智力拼图游戏和迷宫类游戏。这将涉及路径搜索算法，进而引入人工智能这一话题。最终，本书还将探讨某些技术，并将计算机视为问题求解根据，进而搜索数据并构造某些智力游戏。

第 22 章　加 速 方 案

本章包含如下内容：

- 概述。
- 简单和复杂的计算方案。
- 伪物理模拟。
- 剔除操作。

22.1　概　　述

本书重点内容在于代码后的数学和物理知识，通过优化操作，全部代码运行速度均可得到不同程度的提升。本章主要介绍优化操作，并通过预计算值以及空间隔离技术加速代码的运行速度。

22.2　简单和复杂的计算方案

加速代码的核心内容是理解代码中简单或复杂的计算步骤，其中，简单代码占用较少的计算机时钟周期，而复杂代码则消耗大量的计算机时钟周期。相应地，所占时钟周期越长，则处理过程越耗时。本章先期讨论计算速度的测算方式，并通过简单的查找表方式替代某些复杂算法。

22.2.1　计算复杂度

算法的时间长度值称作计算复杂度，并根据函数参数的尺寸加以描述。例如，加 1 操作实际上与数字的尺寸无关，因而下列函数并无实际意义：

```
function sillyAdd(n1, n2)
  repeat for i=1 to n1
    add 1 to n2
  end repeat
  return n2
end function
```

算法的时间长度值约正比于 $n1$ 的尺寸。当该值变得较大时，简单操作（例如存储 $n2$ 并返回结果值）的时间长度值将逐渐变得与此无关。对此，算法将呈现为线性状态且阶数为 1。算法所

包含的既定阶数记为 O(n)，其中，n 表示为阶数。

针对两个数字的加法运算，一类高效的算法则使用第 1 章所讨论的二进制数字。期间，可假设 $n2$ 固定不变，算法占用时间与 $n1$ 的二进制表达位数有关，约正比于该数字的对数，即 O(log(n))。因此，数字越大，与函数 sillyAdd()相比，算法的运算速度也就越快。

下列内容显示了某些创建算法：

- 多项式时间计算。该计算包含 n 次方时间值，前述内容已展示了线性和二次计算，通常情况下，大多数多项式时间算法均与此类似。多项式计算比率与问题的粒度大小有关。若考虑到机器级别的操作，则二次问题有可能转化为三次问题。
- 指数时间计算。该时间值正比于 e^n 并应全力避免——当 n 增加时，算法将变得异常缓慢，例如基于单词表的一类递归填字游戏，第 26 章将进一步讨论此类算法。
- 对数时间计算。对数和多项式时间可组合出现，例如长乘法算法，其阶数为 $n$log(n)。与多项式时间复杂度相比，对数时间则更为快速，因而值得对此进行深究。
- 常数时间计算。该结果可视为算法编制者所追求的目标，此类算法较为稀少，其中的一个示例是两个链表的连接计算。链表可视为数据链，各成员包含自身信息以及指向下一个链接的指针。当对链表执行连接操作时，可简单地将第一个链表的最后一个链接连接至第二个链表的首个链接处。

需要注意的是，对于函数中的较小值，对数的底数较为重要；而处理较大数字时，底数也可能与此无关。另外，数学应用可能无法处理具体的计算操作。在实际操作过程中，较小数字与较大数字间的操作具有类似性。例如，若函数正比于 1000000n，并将其与另一数字（该数字正比于 n^2）进行比较，则计算效率变得尤为重要。对于较小数字，算法的效率则有可能不再重要。

计算复杂度同样适用于基准测试，其中，特定计算采用不同方法运行多次，进而对计算速度进行测量。这里，基准数据须谨慎设置，并置于当前计算环境中。毕竟，连续执行同一计算百万次并不常见。通常情况下，计算过程作为较大处理过程中的部分内容，可对不同的算法值产生显著的影响。例如，某一处理步骤较为快速，但占用较大的内存空间。当某一加速计算方式使用查找表时，即会出现此类情形。而另一个处理步骤耗时稍长，但却生成多个其他中间值以供后续操作使用，进而降低其他处理步骤所消耗的时间。

22.2.2 使用查找表

当处理较为耗时的计算时，查找表不失为一类有效方案。针对既定输入范围，查找表可视为经预计算的既定函数值列表，较为常见的示例则是三角形函数。对此，可不直接计算 sin(x)，并获取表中的最近 x 项以执行插值计算，进而从表中查找 sin(x)。对于三角函数，可通过优化步骤限制数据项的数量，也就是说，仅需通过 0～π/2 范围内的 sin(x)值计算正弦、余弦以及正切数据项列表。除此之外，还可逆向使用查找表以计算反函数。

表中的数据项数量取决于查找所需的精确度以及插值方案。线性插值较快，但在执行三次插值计算时往往会占用较大的存储空间，例如第 10 章所讨论的样条——该操作使用较少的数据点，但处理过程相对繁琐。因此，不同的处理过程其差异十分明显。若数据点不超过 15 个控制点，则可使用三次插值计算 sin(x)，并将精确度保留至小数点第 4 位（为了节省计算时间，可存储

$15 \times 4 = 60$ 个三次系数）。当采用线性插值时，则需要使用不低于 200 个点方可获得相同的精确度。对于三次插值，200 个数据点可将精确度保留至小数点第 7 位。

虽然查找表方案可节省一定时间，但引擎通常使用自身查找表计算此类值。若计算具有较高精度的三角函数、对数函数以及指数函数，使用内嵌计算则可获得较快的计算速度。需要说明的是，仅当“释放”某些数据信息时，查找表方案方为有效。对于大量信息，则可插值步骤，并简单地获取查找表中的最近点即可。

除了计算标准值之外，还可针对特定功能使用定制查找表，例如某一游戏角色或跳跃动作。针对某一平台游戏中的角色动作，无须通过角色每次跳跃时的弹跳物理行为构造其动作，相反，这里可预计算高度表。除了计算速度这一优势外，该方案还可在不同时间和平台上标准化动作行为。

22.2.3　整数计算

在第 1 章曾有所提及，与浮点值相比，整数的计算速度明显占据优势。出于简单考量，尽管该问题常被忽略，但利用整数替换浮点数确实可改善算法的计算速度。这里，可用 1～1000 之间的整数值替换原 0～1 之间的浮点值，并于随后对计算结果进行适当的缩放操作。

然而，整数缩放方案也存在一定问题，这一问题源自舍入误差。当采用浮点数执行多次计算后，积累误差十分明显，因而需要对精确度进行维护。通常情况下，精确度维护算法应依赖于初始数据值，且不应对中间值构造计算。

图 22.1 由一系列均匀排列的正方形构成，并假设沿正方形网格从 A 移至 B，此处需要计算最为接近直线形式的一组正方形。该问题多见于绘图软件中，其中，直线以像素序列加以绘制。

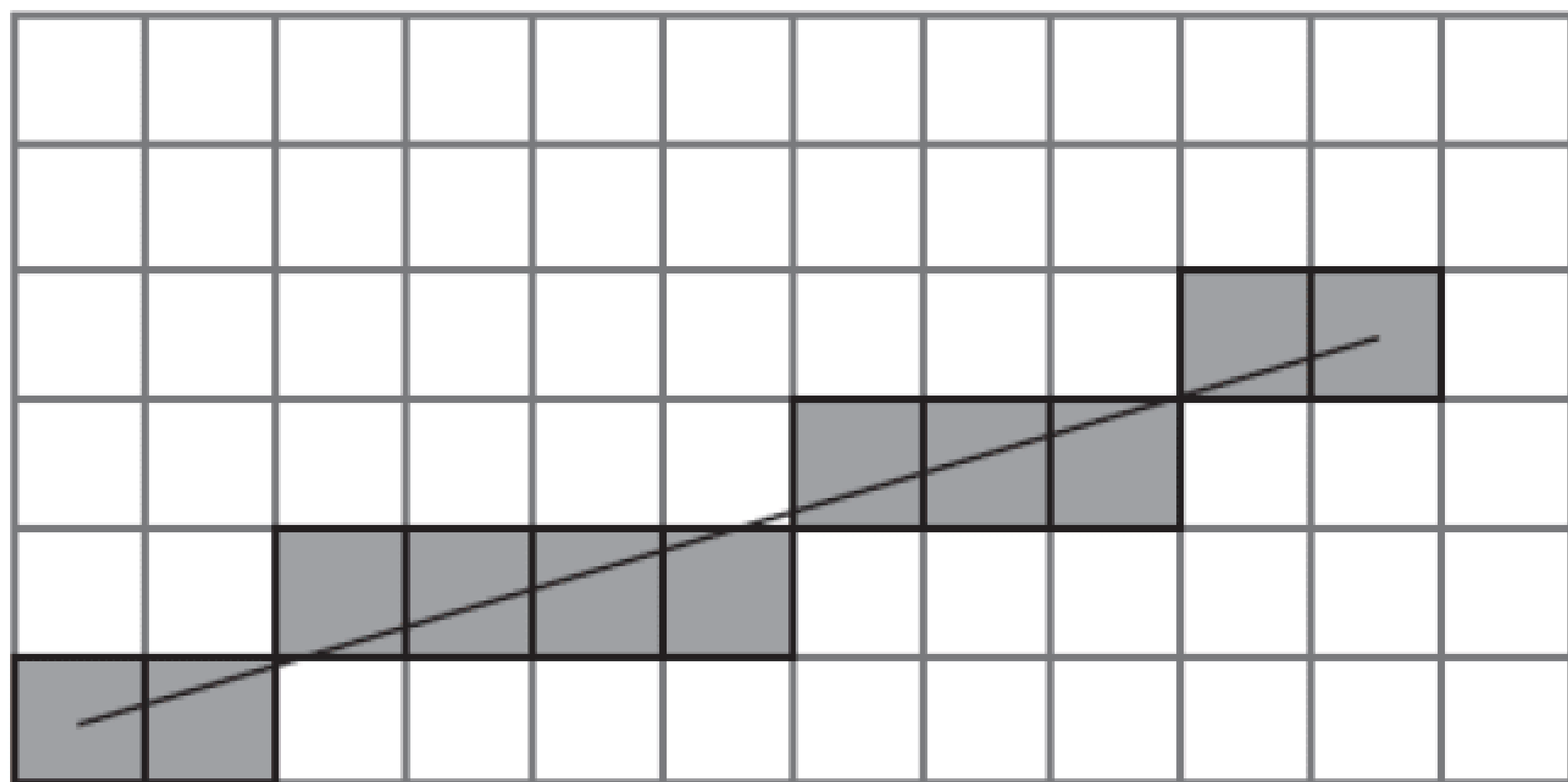


图 22.1　正方形网格中的近似直线

处理网格问题的最佳方案是 Bresenham 算法，图 22.2 显示了该算法的工作方式，并在点 $P_1 = (x_1, y_1)$和 $P_2 = (x_2, y_2)$之间绘制一条直线。出于讨论目的，此处假设 $x_2 > x_1$ 且全部值均为整数，直线的梯度 m 位于 0～1 之间且 y 值沿向下方向计算（最终还将对上述假设条件进行适当调整）。

图 22.2 着重强调了直线的开始阶段。由于前述直线梯度的假设条件，位于点 A 处的像素被填充，下一个目标点为右侧（R）像素或对角线上方（D）像素，其填充方式取决于直线的斜率。特别地，此处须关注位于点 $x_1 + 1$ 上方的 P_1 点位于当前直线（位于 R 和 D 之间）的上方或下方。

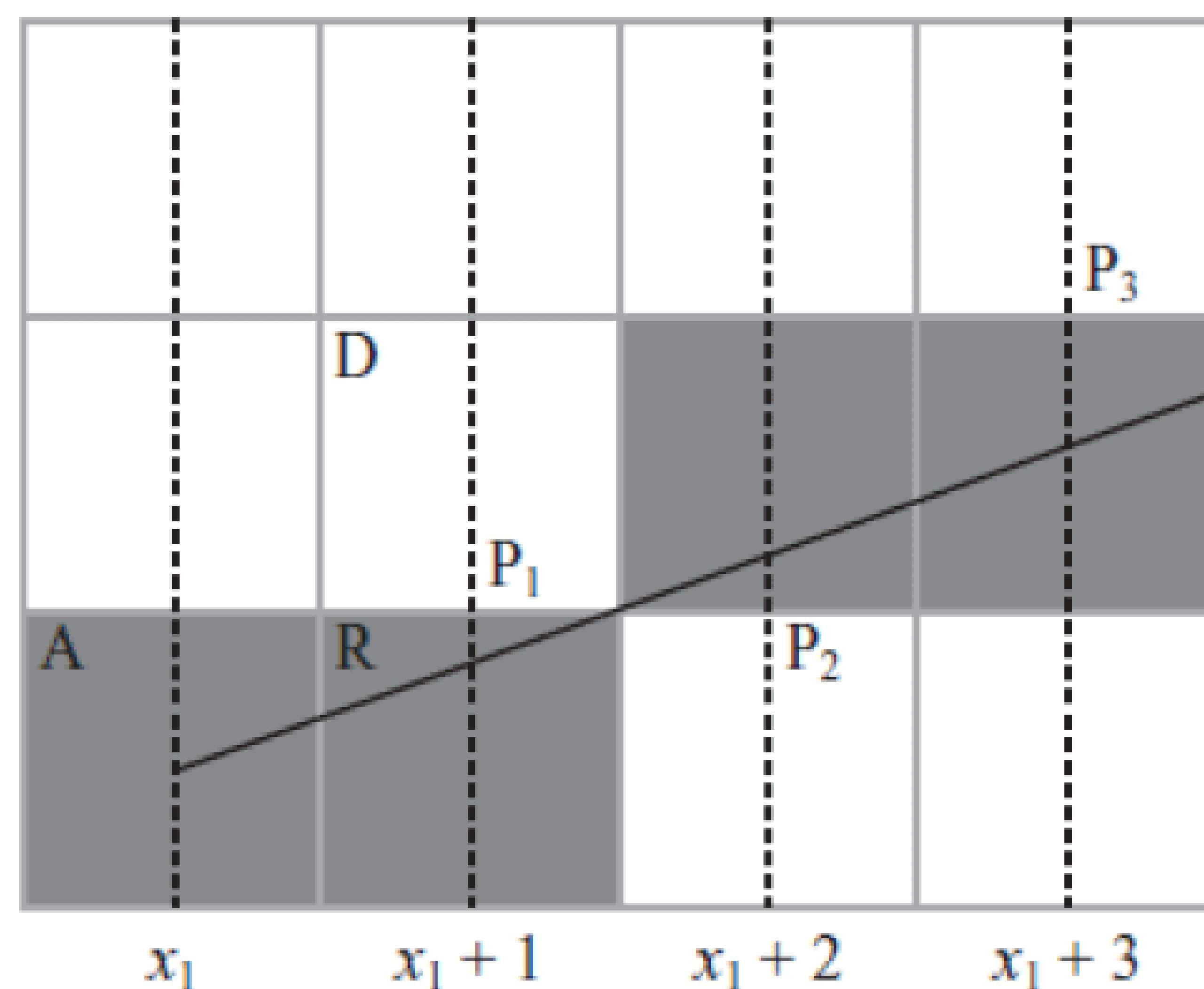


图 22.2　Bresenham 算法

该过程持续进行，进而逐步填充正方形，并在各点处向右或以对角线方式移动（取决于中点处的直线位于半途直线的上方或下方）。在图 22.2 中，即将填充的正方形采用灰色着色，全部工作须在各步骤中以高效方式计算直线位于测试位置的上方或下方，且该过程仅涉及整数数据。此处，可将 m 重记为$\frac{a}{b}$，其中，$a = y_2 - y_1$ 且 $b = x_2 - x_1$。需要注意的是，a 和 b 皆为整数。因此，直线方程 $y = mx + c$ 演变为 $by - ax - bc = 0$。针对 bc，将 A 的坐标代入至当前方程中，有 $bc - by_1 - ax_1$。据此，可定义函数 $L(x, y) = by - ax - bc$。针对直线上方的任意点(x, y)，有 $L(x, y) < 0$；而对直线下方的任意点，则有 $L(x, y) > 0$。

当前，可制定相关迭代方案以处理上述问题。迭代方案与递归方案相反，并定义两个元素。其中，首个元素对应第一个步骤执行的操作；而第二个元素则体现了特定步骤与后续步骤间的数据获取方式。例如，针对一段阶梯的攀爬动作，迭代算法可描述为：首先位于阶梯的底端；随后，各步骤依次到达邻接台阶，最终到达顶端。对此，可创建一系列的中点 P_1，P_2，p_1，…，且各点均通过前一点加以确定。对于各中点，可根据 $L(P_i)$值计算 $L(P_{i+1})$值。

下面考察第一个步骤，即对应角色从(x_1, y_1)处移动，此时，目标中点表示为 $P_1=(x_1+1, y_1 - \frac{1}{2})$，进而判断是否 $L(P1)=by_1 - \frac{b}{2} - ax_1 - a - bc<0$。代入 bc 值后，则有 $L(P_1)=by_1 - \frac{b}{2}$。为了保持整数特征，可将不等式两侧乘以 2，且不会改变最终计算结果。若不等式为真，则可按照对角线方式移动；否则，对应位置则移至右侧——该操作可用于后续各步骤中。在图 22.2 中，实际情况可分为两种情形：若在前一步骤中右移，则目标中点与前一步骤具有相同的 y 坐标，且 x 坐标加 1。若以对角线方式移动，则 y 坐标减 1。

在第一种情形中，若检测到的最后一个中点包含坐标(x_i, y_i)且右移，则可知 $P_{i+1}= (x_i + 1, y_i)$，因而若 $L(x_i + 1, y_i) < 0$，则可再次右移；否则，则按照对角线方式移动。这将生成不等式 $by_i - ax_i - a - bc < 0$，并与 $L(x_i, y_i) - a < 0$ 等同。在第二种情形中，待以对角线方式移动后，即 $P_{i+1} = (x_i + 1, y_i - 1)$，则有 $by_i - b - ax_i - a - bc < 0$ 或 $L(x_i, y_i) - a - b < 0$。再次强调，可在不等式两侧乘以 2，以确保当前计算处于整数范围内。

drawBresenham()函数封装了上述操作，如下所示：

```
function drawBresenham(startCoords, endCoords)
  drawPixel(startCoords)
  set x to startCoords[1]
  set y to startCoords[2]
  set a to endCoords[1]-x
  set b to endCoords[2]-y
  set d to 2*(a+b)
  set e to 2*a
  //calculate 2L(P1)
  set linefn to -2*a-b
  //perform iteration
  repeat for x=x+1 to endcoords[1]
    if linefn<0 then //move diagonally
      subtract 1 from y
      subtract d from linefn
    otherwise //move right
      add e to linefn
    end if
    drawPixel(x,y)
  end repeat
end function
```

drawBresenham()函数所示代码须可处理其他情形，其中，梯度绝对值大于 1。对此，可在算法中交换 x 和 y。对于其他情形，梯度值可为正值，并在各步骤中向 y 坐标加 1（而非减 1）并改变 a 的符号。若初始点的 x 坐标大于端点的 x 坐标，则仅交换两点即可。读者可尝试练习 22.1 以处理此类情况。

Bresenham 算法以及 drawBresenham()函数提供了一类较好的解决方案，进而可理解整数计算的核心内容。除此之外，此类方案还在原有信息基础上进一步展示了高效的整数近似处理方法。Bresenham 算法不仅应用于绘制操作，同时还适用于路径搜索、移动以及碰撞检测中，并以快速、易用的方式，通过丰富的直线和角度对场景世界实现数字化操作。

22.3　伪物理模拟

待考察了运动简化机制并生成快速路径后，还应进一步获取相应的优化方案，即伪物理模拟。伪物理模拟通过简化或模拟可具有真实的运动效果。总体而言，伪物理模拟所提供的问题处理方法均称作近似方案。

22.3.1　对碰撞执行简化计算

如前所述，碰撞计算较为复杂，即使简单的问题亦是如此，例如计算两个圆或圆和旋转直线之间的碰撞点，并涉及数值处理这一类计算密集型方案。

若时间片足够小，则可通过简单的处理过程并在可接受范围内模拟碰撞行为，其中包括两个

步骤：首先计算碰撞是否出现于某一特定时间片内；其次，还可对碰撞结果予以猜测。多数时候，该处理易于实现并可生成令人信服的运动效果。尽管存在某些潜在缺陷，但通常可对相关问题进行处理。

作为一个简单的示例，下面考察两个椭圆之间的碰撞。这里，两个椭圆间的碰撞可转换为某一椭圆沿 x 轴与另一单位圆之间的碰撞行为。也就是说，椭圆 $E(\mathbf{p}, (1\ 0)^T, a, b)$沿偏移向量 $\mathbf{v}$ 移动，进而查看与圆 C(0,1)间的碰撞结果。

若时间片足够小，且在此期间产生碰撞，则椭圆定在时段结束时刻与圆相交。否则，唯一可能的碰撞方式可描述为：椭圆以一个较小的角度掠过圆形。鉴于此类碰撞效果并不明显，因而该角度可忽略不计。

（伪）碰撞计算实际等同于计算椭圆 $E'(\mathbf{p}+\mathbf{v}, (1\ 0)^T, a, b)$与 C 是否相交。换言之，在与原点距离小于 1 的位置处计算 E'上的一点，如下所示：

$$(a\sin\theta+q_1)^2+(\cos\theta+q_2)^2<1$$

其中，$\mathbf{q}=\mathbf{p}+\mathbf{v}$。对此，可计算上式的最小值。通过微分计算并将导数设置为 0 即可得到最小值，如下所示：

$$2a(a\sin\theta+q_1)\cos\theta-2b(b\cos\theta+q_2)\sin\theta=0$$

$$a\sin\theta+q_1=\frac{b}{a}(b\cos\theta+q_2)\tan\theta$$

将上式代入不等式中，则可得到：

$$\left(\frac{b}{a}(b\cos\theta+q_2)\tan\theta\right)^2+(b\cos\theta+q_2)^2<1$$

$$\left(\frac{b^2}{a^2}\tan^2\theta+1\right)(b\cos\theta+q_2)^2<1$$

由于 $\tan^2\theta=\dfrac{1-\cos^2\theta}{\cos^2\theta}$，经过适当的代数调整后，则可得到下列 4 次不等式：

$$b^2d^2c^4+2bd^2q_2c^3+(b^4+q_2d^2-a^2)c^2+2b^3q_2c+b^2q_2^2<0$$

其中，$c=\cos\theta$。此处可通过代数方式求解不等式，若 c 值位于−1 和 1 之间，则产生碰撞。

当处理旋转运动时，也可采用类似的简化方式。例如，包含侧旋的两个运动对象间的碰撞通常可忽略旋转行为。在各处理阶段，可计算对象当前朝向间的线性碰撞。然而，该方法并不能正确工作，无论对象是否处于线性运动状态。类似地，该处理过程还会遗失某些不易察觉的边缘处的碰撞。

22.3.2　简化运动行为

伪物理模拟可用于处理多种运动类型，其中一种即为摩擦力。在前述章节讨论的撞球游戏中，即实现了简单的伪摩擦力系统。其中，球体以恒定速率减速运动。作为一般规则，可根据适应程度对运动行为进行适当简化。对于直觉较为敏感的人士，则需要相对准确地实现模拟过程中所涉及的弹跳行为。另外，大多数人对于振荡行为的认知并不敏感，如果模拟结果近似可行，则无须过分强调此类运动行为。针对均匀的摩擦减速，弹簧伸展、组合时需要使用均匀的摩擦减速。

类似地，人们通常对线性动量和能量易于理解，却对角运动缺乏直观的认识。据此可推测，

人们对旋转现象稍感奇特，因而过度简化角运动将带来一定风险。例如，可在侧旋方向上施加额外的冲量以处理侧旋碰撞，而非精确地计算角分量和线性分量。

在某些游戏环境中，与精准的物理模拟相比，人们更乐于接受伪物理模拟。在赛车类游戏中，若游戏采用真实的物理学内容驾驶赛车或四驱车，则与简化后的离散运动相比，车辆将变得难以控制，对应运动包含前、后、左、右等运动行为。针对游戏体验，游戏制作者应判断是否有必要将实时物理行为引入至游戏中，进而提升游戏的难度。练习 22.2 将进一步讨论这一问题。

22.3 剔除操作

在讨论碰撞和可见性计算时，曾多次提及剔除操作，但未曾对其进行深入讨论。剔除操作可快速消除特定问题中的非候选对象，例如当确定场景可见部分时相机后方的相关对象。这里，存在多种技术可有效地改善剔除操作，且多数隶属于划分树这一范畴。

22.3.1 空间划分

在第 11 章中，曾将游戏场景划分或分割为多个小型块状对象，该技术可通过两种方法实现。方法一是制定较小的块状对象，并将其置入最终的游戏场景中，即第 23 章所讨论的拼贴型游戏。方法二则是执行递归处理，进而将场景世界按照多个嵌套区域方式进行组织，并一次构造划分树。本节将讨论划分树的各种生成方法，且适用于二维和三维场景。相关示例主要集中于 2D 维度，而某些 3D 游戏实际上通过 2D 或 2.5D 方式进行制作。

划分树可视为一类数据结构，并以层次结构存储场景世界信息。实际上，树形结构等同于第 18 章使用的系统，用于处理父子关系转换并包含大量的节点。其中，全部节点包含一个父节点以及 0 个或多个子节点。若某一节点不包含子节点，则该节点称作叶节点；而最上方节点不存在父节点，因而称作根节点。据此可知，树形结构通常以上下颠倒方式予以绘制，且树形结构通常不包含环路。另外，不存在节点为其自身的父节点、子孙节点等其他相对关系。这里，计算信息与各个节点关联，对此，读者需要深入理解面向对象的编程理念。

【提示】树形结构是图结构的一个特例，这种关系将在第 24 章加以讨论。

在划分树中，根节点体现了全部场景世界，并可根据某一方案划分为较小的部分（通常情况下彼此不交叠）。随后，可对各部分内容再次划分，该过程重复执行，直至到达某一预置条件，例如最小尺寸值。其中，划分方式基于某种简化计算，例如光线跟踪或抛掷检测。特别地，若父节点无效（包括可见性以及邻接特征等），则该父节点的子节点也将处于失效状态。这也意味着，对于与父节点关联的树形分支，无须对其执行进一步检测。

在第 11 章曾谈到，划分树并不能解决全部问题。例如，父节点自身的检测将增加额外的计算。若计划检测场景世界中的全部对象，可见性剔除操作毫无用途。类似地，撞球游戏中的一次击球后，则桌面上的各球体可能均会受到影响，因而需要对其进行检测。无论如何，在大多数场合下，划分树均可视为一类较为有效的技术；而在某些特殊环境中，例如游戏场景中包含了大量

的室外地形，该技术不可或缺。

22.3.2　四叉树和八叉树

四叉树是划分树中一种较为简单的形式，其 3D 对应结构为八叉树。针对撞球游戏中的系统，四叉树可视为一种最为自然的扩展方式，其中，场景被划分为多个正方形区域。这里，假设某一较大的正方形涵盖了全部 2D 平面，并将其作为根节点。随后，可将根节点划分为 4 个正方形，即当前根节点的子节点，4 个子节点再次划分为较小的正方形。如图 22.3 所示，子节点的划分处理持续进行，直至正方形到达预设最小尺寸。

与各个叶节点关联的数据表示为节点区域所含的对象集合，包括全部几何对象（例如球体）或者独立的多边形直线段（3D 网格将使用到多个多边形）。多个正方形内的对象将与全部叶节点关联；与各个父节点关联的数据表示为平面内的顶点集合，也可能是其所包含的全部对象复制集合。

相应地，存在多种树形搜索算法可用于处理各类问题，例如碰撞检测。当检测处于碰撞状态的对象时，针对各对象，若父节点未包含既定时间片内的对象轨迹，则可对其予以剔除。如图 22.4 所示，可先期剔除深灰色区域；随后，可剔除次级灰色的区域。该过程持续进行，直至获得与运动轨迹相交的叶节点集合。最后，可快速对叶节点内的对象执行碰撞检测。

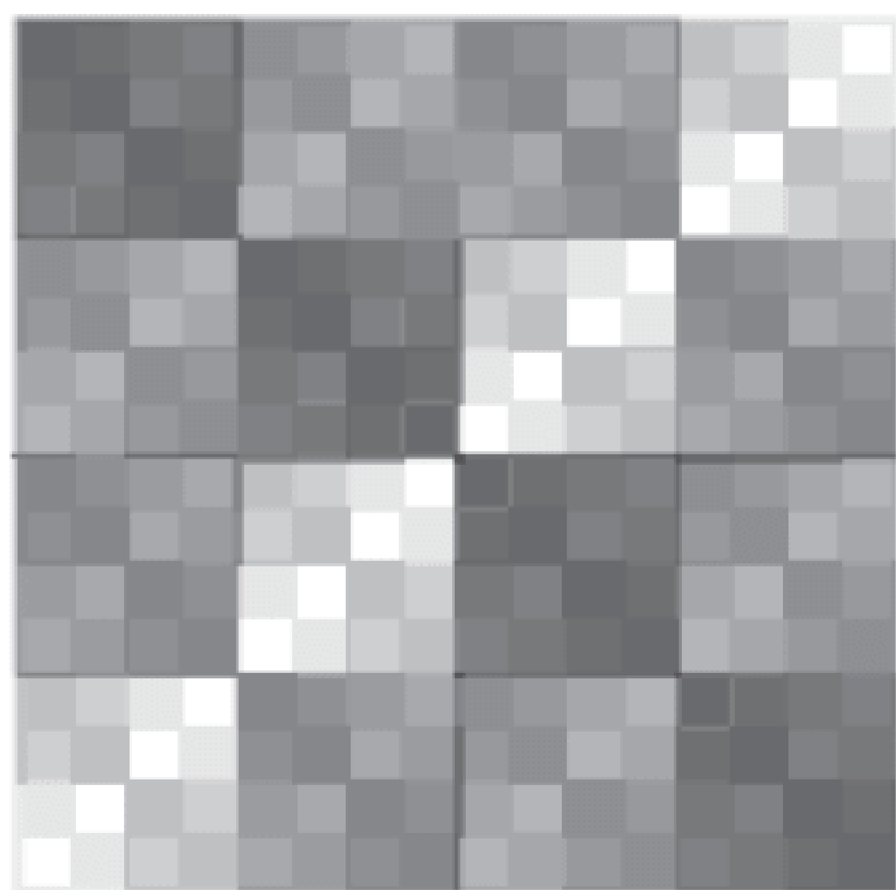


图 22.3　基于四叉树的平面结构

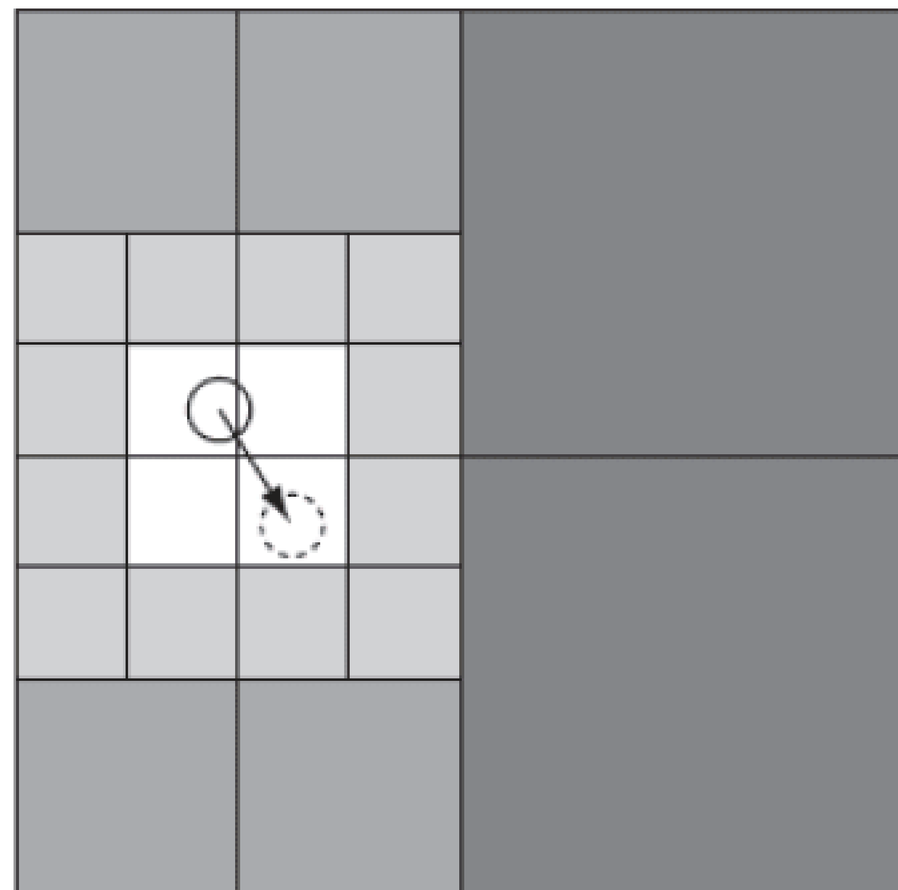


图 22.4　基于四叉树的碰撞检测

尽管上述算法并不复杂，但其速度并不快于将空间划分为平面网格。在某些时候，其速度甚至更慢。然而，为了加速计算过程，针对既定叶节点中的对象，可事前计算节点集并可实现自动剔除。如果运动量相对于网格尺寸较小，则假设第一级划分的中间对象无法到达其他一级节点。尽管该方案通常较为高效，但并不适用于特定父节点的边缘节点。

四叉树常用于可见性判断，并可减少计算过程中所需的叶节点数量。除此之外，四叉树还可用于光线跟踪计算中。据此，可得到与光线相交的父节点集合。若已知父节点为空节点，则可对此予以忽略；否则，还需进一步对子节点执行递归操作。一类值得深究的算法是 Bresenham 算法，并可通过快速的整数运算执行上述操作。

在对象级别，四叉树可用于存储碰撞图。图 22.5 显示了一类碰撞图，并通过如下算法划分为四叉树：若特定区域全空或全满（例如划分 A 和 B），则可停止树形结构的子划分；否则，须

对子节点进行递归操作。同时，还可对算法执行进一步的优化操作，也就是说，当特定节点包含某一直边时，算法即可停止。当检测到此类形状间的碰撞后，则可遍历树形节点，直至检测到某一碰撞，或者该级别的全部叶节点为空。

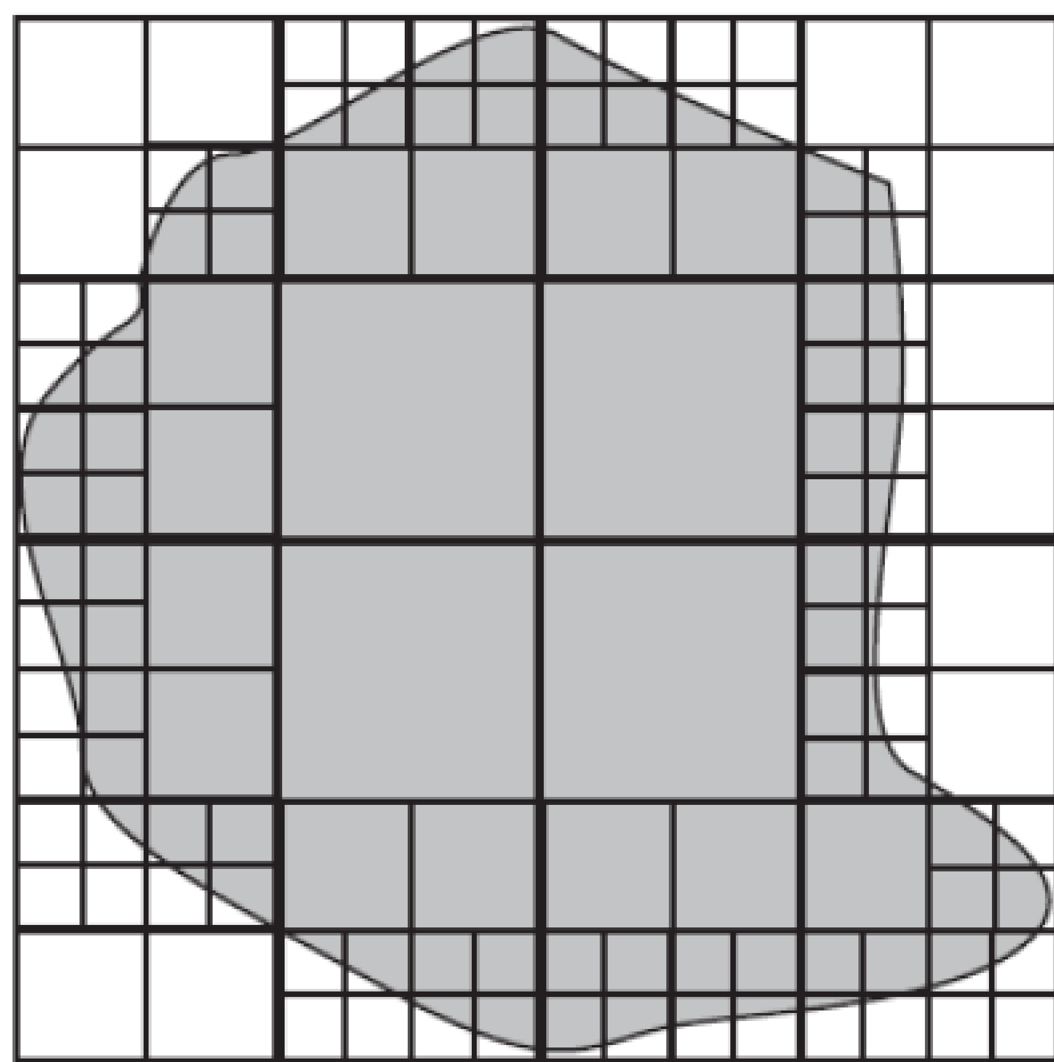


图 22.5　按四叉树方式组织的碰撞图

22.3.3　二分空间

二分空间划分树表示为第二种划分类型。与四叉树不同，BSP 树中的各个非叶节点包含两个子节点，因而各阶段通过 2D 环境中的直线或 3D 中的平面，将每个节点划分为两部分内容。在图 22.6 中，叶节点包含某一完整对象，该对象可能为直线或多边形。分割线常选取为某一特定朝向的直线段或多边形。这也意味着，当对象移出当前位置后，须对 BSP 树重新计算。因此，此类树形结构适用于静态场景，BSP 树中的各节点记录其内的全部对象信息。

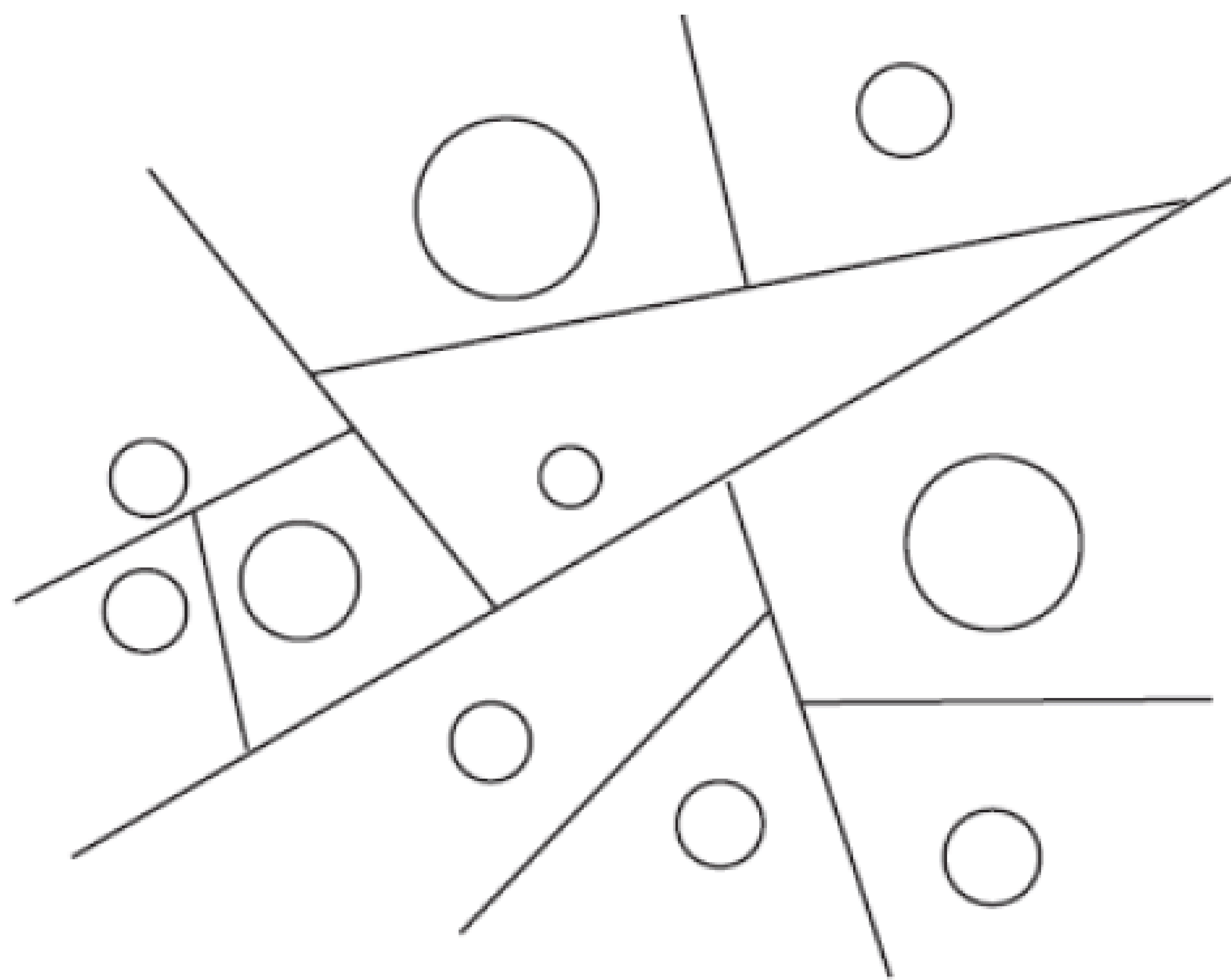


图 22.6　平面的二分操作

在 BSP 树中，全部区域皆为凸体，这也是这一类树形结构的一个重要特征。据此，BSP 树提供了一类高效的方式，进而将凹体对象划分为凸体组件。类似于四叉树，BSP 也适用于场景关卡，并可与可见性和光线跟踪计算协同使用。在对象级别，还可用于检测对象间的碰撞行为。

22.3.4　包围体层次结构

包围体层次结构或 BVH 与当前所涉及的空间划分概念具有相反含义，BVH 关注点在于对象及其基于包围形状的组织方式，而非区间，且常用于复杂的 3D 场景中（同样适用于 2D 环境）。相应地，包围体涵盖球体、AABB、OABB 以及其他形状。

【提示】对象对齐包围盒简称为 OABB；AABB 则是轴对齐包围盒的简称。相比较而言，AABB 更快速，但准确度稍差。

BVH 通常并不包含全部游戏场景世界，其原因在于，包围体空间外部可能包含较大的空区域；另外，不同空间体之间也可能彼此交叠。图 22.7 显示了 2D 包围体层次结构，其包围形状为圆形。

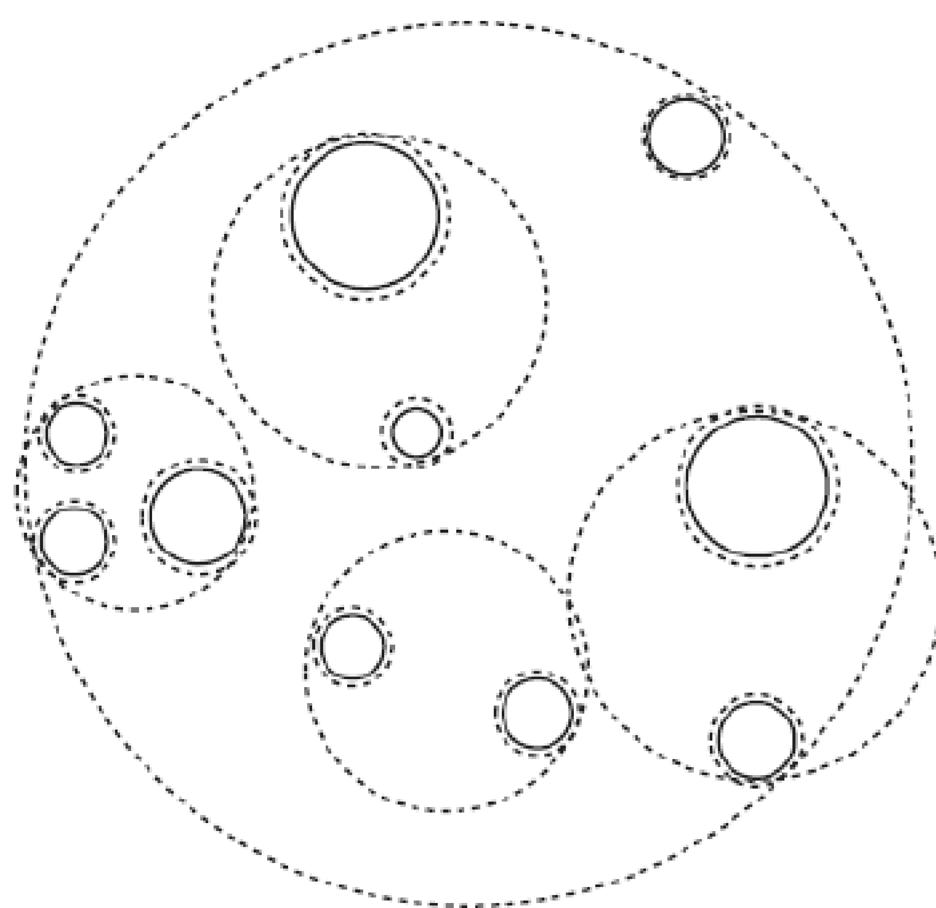


图 22.7　基于分离空间的 2D 包围区域层次结构

对于剔除操作而言，BVH 十分有效且优于其他系统，特别是由诸多动态对象构成的游戏场景，并可通过多种方式与空间划分机制结合使用。

BVH 的难点在于先期阶段构造树形结构。出于效率考量，在树形各级结构中，应使包围体空间尽可能地处于紧凑状态。取决于游戏的构建方式，用户可选取不同的系统。特别地，彼此邻近的对象应以分组方式加以组织。在对象内部，同样可构建包围体层次结构，并近似包含对象形状。图 22.8 显示了小型 OABB 层次结构所包围的对象。

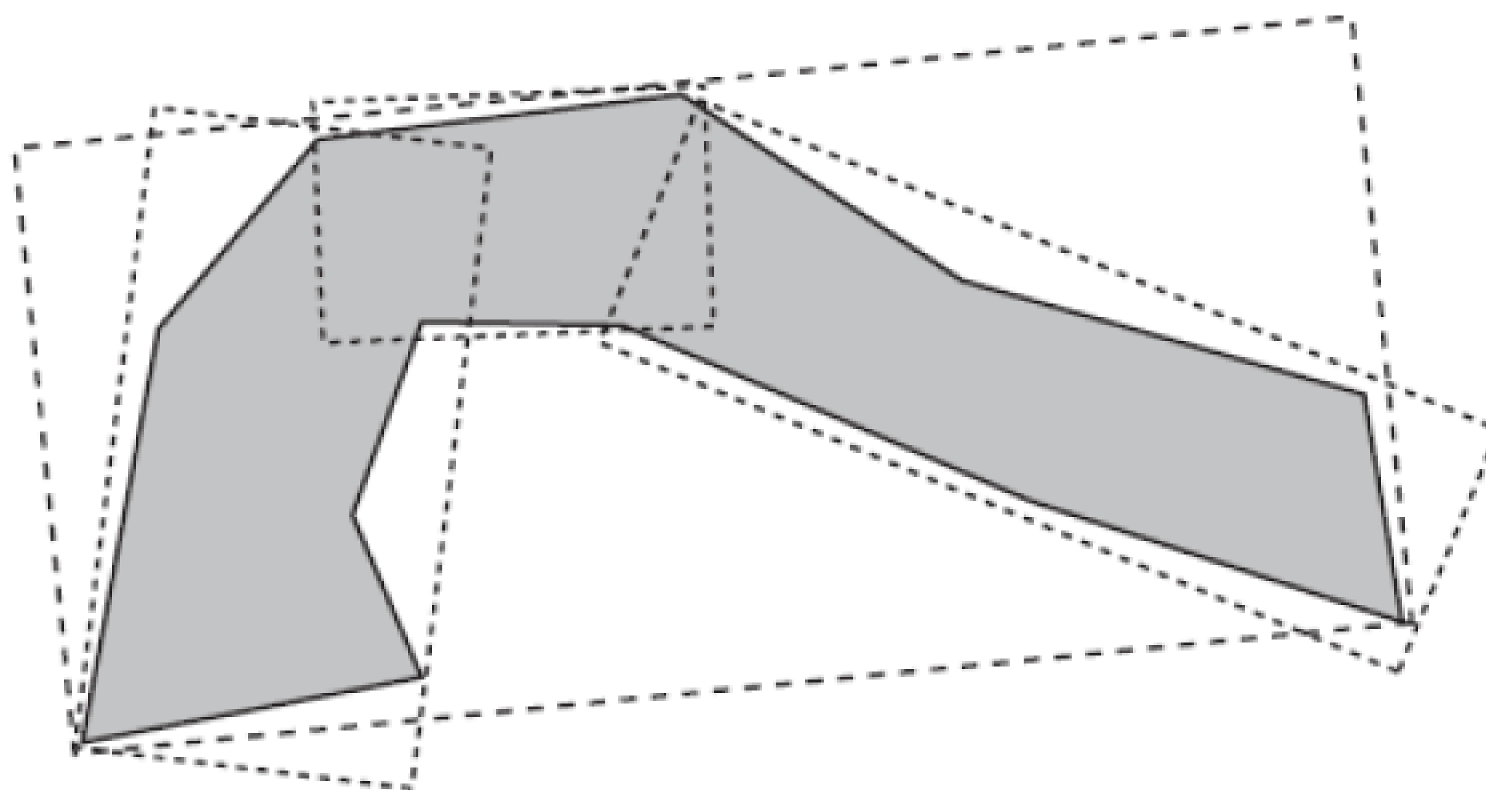


图 22.8　通过 BVH 简化形状几何体

当对象相对于另一对象显著移动时，需要生成更多的 BVH。更多的 BVH 涉及快速分割、合并以及插入树形中的新节点。该过程较为复杂，并涵盖了多种处理方案。当处理大型的复杂场景并对树形系统进一步划分时，相关方案则值得深究。

除了前述剔除系统之外，其他技术还包括可视图方案，并存储可见区域中的细节内容。读者可参考附录 D 以获取更多内容。

22.4 本 章 练 习

【练习 22.1】扩展前述 drawBresenham()函数，以使其可处理各种可能的梯度值，函数注释中包含了相关提示。

【练习 22.2】试编写函数并通过简化版的物理知识以及鼠标键控制车辆的运动行为。在大多数游戏中，标准的车辆控制系统多采用上、下箭头实现加速或减速运动；除此之外，还可通过左、右箭头实现转向操作。例如，当倒车时，左向旋转则使得前向方向指向右侧。另外，若垂直于前向方向的动量高于某一阈值，则转向功能将被锁定。

22.5 本 章 小 结

鉴于篇幅有限，本章仅讨论了游戏设计中的少量核心内容，并希望能够引起读者足够的重视。另外，读者还可参考附录 D 以获取更多内容，旨在开阔读者的阅读思路。本章特别强调了相关计算的简化和优化方式，并简要讨论了剔除操作和游戏场景数据的优化计算。第 23 章将讨论与游戏场景匹配的划分结构。

至此，读者应掌握如下内容：

- 如何通过 O(n)标记计算算法的复杂度。
- 如何通过 Bresenham 算法计算路径。
- 如何使用近似方案生成模拟运动以及碰撞行为。
- 如何使用四叉树、BSP 树以及 BVH 加速可见性以及碰撞计算。

第 23 章　贴 图 游 戏

本章包含如下内容：

- 概述。
- 根据位数据创建游戏。
- 基本的运动和相机控制。
- 高级贴图机制。

23.1　概　　述

益智类休闲游戏可视为一种常见的游戏类型，其中，角色需要与场景世界进行对话。其中，游戏场景由多个障碍构成并置于不同的关卡中，以使玩家完成最终的游戏任务。此类风格多为 2D 游戏，例如 Super Mario World，以及具有少量 3D 特征的冒险类游戏，例如 Tomb Raider，Sonic 以及 Sim 系列。在最初版本中，全部游戏均遵循相同的理念而设计。若读者访问 Kongregate.com 网站，将会发现此类传统风格仍出现于当今的在线游戏中。同时，网站中涵盖了大量 Flash 游戏，其数量仍处于增长状态。对于大量的益智类休闲游戏，其场景往往由贴图或小型的独立单元格构成，并包含多种优点。例如简单的关卡设计、图像复用时较低的内存开销、交互和漫游操作的多样性，以及碰撞检测的便捷性。尽管某些话题超出了本书的讨论范围，但最后两个问题依然需要引起足够的重视，本章将对此予以讨论。

23.2　根据位数据创建游戏

贴图机制涉及较为广泛的内容，在开始阶段，本章将考察系统的核心内容，以及游戏的生成、存储以及运行方式。这涉及游戏场景的构造以及如何将其与角色和其他属性进行有机结合。

23.2.1　构造贴图场景

贴图游戏始于简单的关卡编辑器程序，独立的贴图单元将拖入至正方形网格中。其中，关卡设计者负责构造关卡并生成相应的智力游戏。同时，根据游戏的细节内容，关卡设计者将对复杂度进行适当调整，但最终结果并无太多出入，即各正方形被赋予特定的贴图。随后，当加载关卡时，网格数据将被传输至内存中。

各贴图分别包含多种属性，例如，某块地表可处于融化状态、具有较低的摩擦值、严禁触摸（可能导致生命危险）或者具有水面波动特征。然而，与此相比，与贴图相关的碰撞信息则显得更为重要。

在游戏场景中，可于其中创建大量的移动角色、敌方角色以及相关场景内容（某些可处于运动状态，例如浮动的平台），此类对象可显著地提升游戏的观感。对此，可适当放置游戏对象，例如障碍物或标记。尽管此类对象并未与贴图对应，但依然会增加操作的复杂度，因而当与贴图协同工作时，须对此予以谨慎处理。除此之外，还可生成背景图像（源自贴图），并与相机控制协同工作。

关于贴图游戏的开发项目，上述元素可视为游戏引擎的全部内容。综上所述，游戏引擎包含 4 个主要部分，即贴图数据、固定对象、处于运动状态的环境对象以及游戏角色（包括玩家）。在游戏的体验过程中，引擎需要将此类图像合称为屏幕上的独立图像。

23.2.2　基本的运动行为和相机控制

游戏场景中的运动行为源于自身的坐标系，通常称作游戏空间，因而该空间包含了游戏场景。相对于游戏空间的原点，各运动对象包含自身的位置。通常情况下，原点位于网格的左上角。另外，全部碰撞计算均在该网格或坐标系内部进行。同时，屏幕可视为面向游戏场景的视口，独立或半独立相机通常在角色漫游时予以跟随。这里，相机是否独立取决于玩家是否对相机加以控制。在简单的 2D 视角中，可认为相机位于距游戏主平面特定位置处。其中，游戏场景世界中的一个像素可能对应于屏幕上的某一像素。

drawWorld()函数名称同时显示了其功能。当给定特定的相机位置后，该相机将在视口中绘制经选取后的部分游戏场景。此处，假设游戏场景中的相机位置设置为屏幕中部的像素坐标，如下所示：

```
function drawWorld(tileList, cameraX, cameraY, w, h)
  set x to cameraX-w/2
  set y to cameraY-h/2
  set hoffset to (x mod 16) //assuming 16-pixel tiles
  set voffset to (y mod 16)
  set leftmost to 1+(x-hoffset)/16
  set topmost to 1+(y-voffset)/16
  //add error checks to ensure you aren't out of range
  set largeImage to an empty buffer
  repeat for i=0 to w/16
    repeat for j=0 to h/16
      draw tileList[i+leftmost, j+topmost] to
      square(i*16,j*16,(i+1)*16,(j+1)*16) of largeImage
    end repeat
  end repeat
  copy rectangle(hoffset,voffset,hoffset+w,voffset+h) of largeImage to screen
end function
```

尽管 drawWorld()函数相对低效，但依然创建游戏场景的基本特征。此处使用了 16 像素贴图，

进而构建了一个 2D 场景世界。在实际操作中，存在多种方式可对该函数进行优化。例如，对于单一方向上的卷轴游戏（侧向卷轴或上方卷轴），可按逐列或逐行方式绘制场景，进而在某些单元块位于画面之外后向当前图像画面添加新的单元块。该操作可节省合成时间，但在处理多向卷轴时，处理速度相对缓慢。另外，若玩家选择反向行进时，须重复计算图像块，因而其效率较为低下。取决于关卡的尺寸，一类替换方案则可于初始阶段绘制全部关卡，并将其置于内存中的单一图像中。随后，可在必要时将部分图像复制于屏幕上。

若以处理速度为代价，则可添加图像层并生成视差卷轴，进而丰富场景图像的观感。这里，视差卷轴涉及背景图或前景图，此类图像置于距相机一定距离处，并以此区分焦平面。随后，视差场景以不同的速度滚动。作为视差卷轴的一个实现示例，可生成一个背景图像，该图中的一个像素表示场景空间的两个像素。当使用此类方案时，被感知图像其距离 2 倍于焦平面。

待贴图平面绘制完毕后，随后需要绘制其上的运动对象，这需要将对象的位置从世界坐标转换至屏幕坐标。鉴于对象位置基于焦平面，则可从其位置中减去视口左上角位置，进而计算出屏幕位置。在某些时候，对象可能并未置于屏幕上。

最后一个问题是相机的位置。开始时，并不建议将相机直接链接至角色位置处，其原因在于，当角色位于游戏场景世界边缘时，将在边侧生成包含空白区域的图像。因此，当角色靠近边缘区域时，至少应偏置相机位置。

通常，相机可能浮动于角色后方，并表示为真实相机，且通过插值方法围绕于游戏角色附近。例如，当角色以既定方向运动时，相机可稍加滞后。除此之外，还可着重选择角色的某一侧面，进而查看角色的视见方向。例如，在侧向卷轴游戏中，与角色后方相比，玩家更多关注于该角色的前向内容。当角色停止时，玩家并不希望其位于屏幕中心处。另外，若角色观察右侧内容，则其应位于中心左侧位置处。

23.2.3　基本的碰撞行为

待游戏场景制定完毕后，下一项任务即是确定碰撞检测功能。除了内存和速度优势以外，单元贴图的最大优点在于可动态地简化碰撞检测计算。这里，碰撞检测的处理方式取决于地形是否定义为实体，亦或是否具有可穿透性。若二者兼具，则可将上方贴图确定为实体单元，这也是此类游戏的常见设计方式。据此，当角色移动时，则仅当角色部分进入新贴图单元时执行碰撞检测。detectCollisionWithWorld()函数即采用了这一方案，如下所示：

```
detectCollisionWithWorld(c, w, h, displacement, tiles)
  //w and h are the width and height of the box
  //c is the top-left corner

  //determine the colliding edges
  if displacement[1]=0 then set edge1 to "none"
  else if displacement[1]>0 then set edge1 to c[1]+w
  else set edge1 to c[1]

  if displacement[2]=0 then set edge2 to "none"
  else if displacement[2]>0 then set edge2 to c[2]+h
  else set edge2 to c[2]
```

```
//calculate first collision
set t1 to 2 //time to collision along vertical edge
if edge1<>"none" then
  set currTileX to ceil(edge1/16.0)
  set newTileX to ceil((edge1+displacement[1])/16.0)
  if currTileX>newTileX then
    set t1 to ((currTileX-1)*16.0-edge1)/displacement[1]
  otherwise if currTileX<newTileX then
    set t1 to (currTileX*16.0-edge1)/displacement[1]
  end if
end if

set t2 to 2 //time to collision along horizontal edge
if edge2<>"none" then
  set currTileY to ceil(edge2/16.0)
  set newTileY to ceil((edge2+displacement[2])/16.0)
  if currTileY>newTileY then
    set t2 to ((currTileY-1)*16.0-edge2)/displacement[2]
  otherwise if currTileY<newTileY then
    set t2 to (currTileY*16.0-edge2)/displacement[2]
  end if
end if

if min(t1,t2)=2 then return "none" //no change of tile

if t2<t1 then //first collision is along horizontal
  set newTile to newTileY
  set currTile to currTileY
  set checktile to [ceil(c[1]/16.0), ceil((c[1]+w)/16.0)]
  set mx to the number of columns of tiles
  if displacement[2]>0 then set dir to "bottom"
  otherwise set dir to "top"
otherwise
  set newTile to newTileX
  set currTile to currTileX
  set checktile to [ceil(c[2]/16.0), ceil((c[2]+h)/16.0)]
  set mx to the number of rows of tiles
  if displacement[1]>0 then set dir to "right"
  otherwise set dir to "left"
end if
//at the end of the above process, newtile and currtile give the
//changed row or column, checktile gives the start and finish
//of the tiles containing the player

set t to min(t1, t2)

if newTile<1 or newTile>mx then
  //edge of map
  return [c+t*displacement, (0,0),dir]
end if
```

```
  //check whether any new tiles entered are solid
  repeat with i=checktile[1] to checktile[2]
    if dir="bottom" or dir="top"
      then set tile to ptiles[newTile][i]
    otherwise set tile to ptiles[i][newTile]

    if tile is not empty then
      //potential collision
      if tile.solidity="solid" or
         (tile.solidity="top" and dir="bottom") then
        //tile collision (there could be more options here)
        //move to collision point
        return [c+t*displacement, (0,0),dir]
      end if
    end if
  end repeat

  //no collision: recurse
  if t=0 then set t to 0.001
  return detectCollisionWithWorld(c+t*displacement, w, h, (1-t)*displacement)

end function
```

【提示】根据惯例，可方便地将处于运动状态的游戏角色视为轴对齐包围盒，当然，也可将类似方案应用于游戏中的其他对象上。

针对 detectCollisionWithWorld()函数，可使用第 21 章介绍的技术。例如，可以计算角色在一段时间内跳跃的高度值，并将其存储于一个数据表内。又如，为了节省计算时间，还可假设游戏角色位于地面之上。在游戏 Mario 中，角色若位于两个贴图单元的连接处，则需要首先检测是否位于实体地面上——此位置有可能出现碰撞行为；而水平方向上则无须考察碰撞结果。

关于运动行为的控制方式以及碰撞结果，需要注意的是，休闲类游戏很少具有物理真实性。例如，当角色跳跃时可能会改变方向，当与地面碰撞时，该对象无须包含弹跳行为。在第 22 章曾有所提及，此类行为使得游戏更具吸引力。

23.2.4　复杂贴图单元

如前所述，大多数 TBG 均使用实体贴图单元，但也包含其他选项。其中，可使用碰撞图将贴图单元细分为较小的区域，如图 23.1 所示。对此，可使用碰撞图并仅根据某几处的实体贴图单元或空体贴图单元执行碰撞检测（而非通过完全实体/空体贴图单元计算碰撞）。

当使用图 23.1 所示的碰撞图时，可存储法线细节内容以及碰撞信息。由于上述方案在一定程度上增加了碰撞检测的复杂性，因而可适当限制使用过程中贴图单元的数量。除了碰撞图位图之外，还可尝试使用向量方案。对于后者，图 23.1 中的贴图单元，可看成右上方至左下方的“墙面”。尽管缺乏一定的通用性，但该方案可增加碰撞检测的计算速度。

贴图单元于一段时间内的变化可视为一类扩展操作，其变化可呈现为多种方式，例如传送带、

在某种作用下的触发状态以及冰块的融化。从根本上讲，此类变化的实现过程并不复杂，但在合成处理过程中存在一定的复杂性。为了节省计算时间，当特定贴图单元方式变化时，无须重复绘制全部关卡。相反，仅需适当调整当前视野范围内的贴图单元即可。对此，可在现有关卡图像基础上予以绘制。

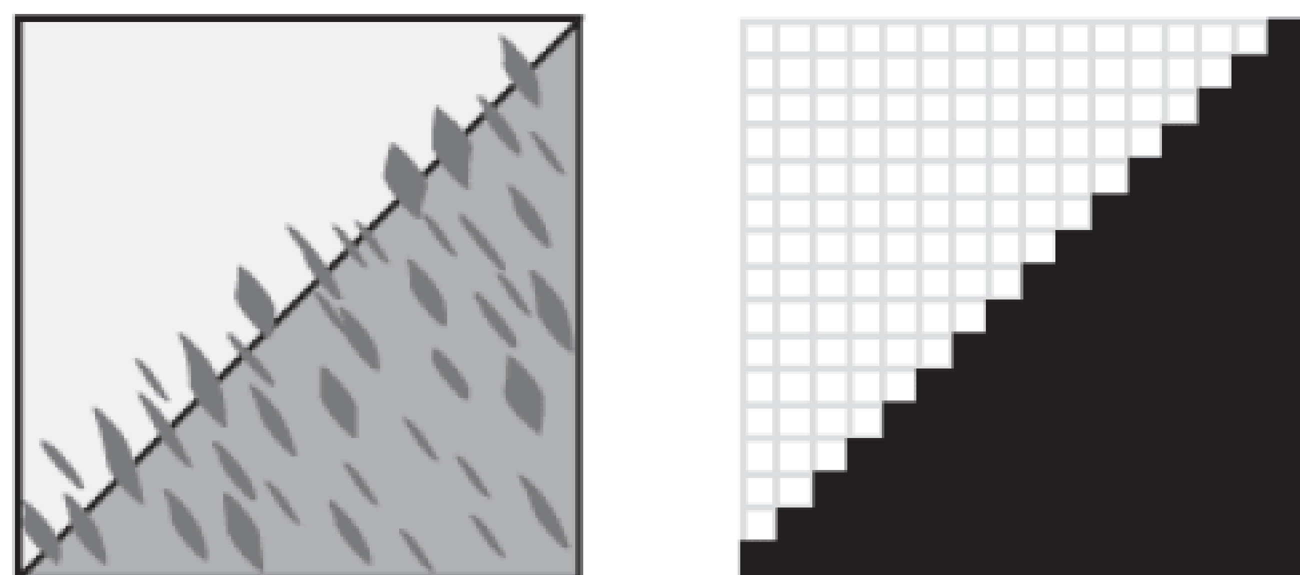


图 23.1　基于关联碰撞图的贴图单元

作为进一步的扩展行为，贴图单元可在不同位置间移动，例如传送带，此类对象可作为游戏角色进行处理，而非关卡贴图单元。待固定场景复制完毕后，这一类贴图单元方可在关卡图像之上予以绘制，例如生长中的树木、爆炸的木桶等，是否将其视为独立对象（绘制于场景上方）亦或是贴图单元则取决于计算量以及优化程度。总体而言，对于贴图单元，为了增加计算效率，可做适当优化，但并非绝对必要。

23.3　高级贴图机制

前述讨论主要集中于 2D TBG 中，但相关技术也适用于 3D 或 2.5D 游戏中，且多数操作产生于平面上，甚至是样条表面上。

23.3.1　等轴测视图

大多数传统意义上的 3D 贴图单元游戏均采用等轴测场景，在第 17 章曾对此有所提及，且与 2D TBG 并无显著差异。如图 23.2 所示，二者的唯一差别在于，体现地面形状的轴测图贴图单元采用菱形绘制，而非正方形，但事实并非全部如此。由于等轴测视图表示为 3D 场景，因而该场景的等轴测贴图单元包含高度、长度以及宽度。总体而言，贴图单元的高度随该贴图单元一同存储。当在适宜位置绘制贴图单元时，常通过存储的高度值进行偏移。

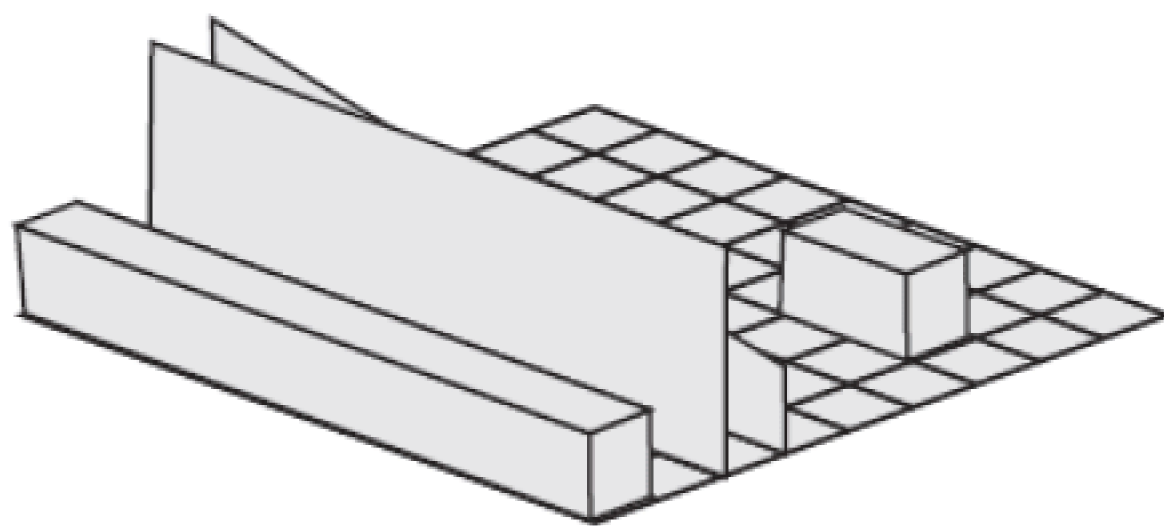


图 23.2　轴测图场景世界

当前，大多数开发人员通过 3D 加速机制显示等轴测场景的诸多特征。然而，此类型游戏的原始版本多采用类似于 2D TBG 方案进行实时绘制。

对于较好的等轴测游戏，其难点在于游戏角色的控制，主要问题可描述为：网格上角色左右、上下移动，但屏幕上的移动无法实现一一对应。对此，相关处理方案也不一而同。对于数据点-单击界面，可通过单击目标位置告知角色移动的具体位置。另一种方法则是使用对角线标记，例如经典的 Qbert 游戏。

一类常见问题常产生于角色以动态方式移动时，例如角色在等轴测地表上方飞行时。若采用 3D 方式计算，则不存在中间方式区分近相机对象以及远距离对象（但存在高低值之分）。由于场景世界表示为等轴测视图，这里甚至无法使用相对尺寸，对此也存在不同的处理方法。一类早期方案则是使用阴影，并以此显示角色与贴图单元平面的相对位置。

23.3.2　3D 贴图类游戏

根据惯例，等轴测 3D 的使用并非硬性规定。同一贴图单元技术同样适用于 3D 场景，例如游戏 Tomb Raider，该游戏可清晰地看到原始的贴图单元。大多数游戏场景多由不同尺寸和形状的正方形块构成。尽管如此，游戏 Tomb Raider 和 Super Mario World 之间仍存在一定差别。二者的核心内容可描述为：角色在贴图单元场景环境中搜寻目标，躲避或击败敌方角色，并在沿途收集各种道具。

【提示】需要注意的是，并非全部 3D 游戏均采用贴图单元。例如，Doom 和 Quake 引擎则使用了压制（extrusion）系统。在该系统中，墙面的绘制与 2D 贴图保持一致。

多数与 3D 等轴测游戏相关的问题常与相机控制有关，第 24 章将会对此进行深入讨论。若缺乏对相机的控制，当前类型游戏将会遇到与 TBG 相同的问题。对此，前述解决方案依然有效。然而，由于维度的增加，对应技术难度也有所提升。例如，对于贴图单元，由于无须遵从等轴测网格，因而若采用 3D 场景，则贴图机制将会涉及更多有趣的形状，此类形状可能包含斜面上端。另外，贴图机制还可能在表面使用纹理和光照，且独立于贴图单元形状，这将为前期设计留有更多余地。贴图单元的碰撞和视效元素也将予以独立计算。

针对碰撞检测技术，类似于前述章节讨论的复杂贴图单元机制，3D 环境提供了某些更为有趣的选项。其中，可通过高度图定义贴图单元，也可采用向量描述方法。例如，从高度为 50 处自左向右下斜 30°。若贴图单元较小，使用向量描述不失为一种较好的选择。此外，该方案还可进一步构造具有丰富内容的表面结构，当使用纹理和凹凸贴图时尤其如此。

相应地，可将上述讨论结果应用于代码中，box3DTileTopCollision()函数即使用了向量方案，进而测试三维 AABB 与单一贴图单元顶端之间的碰撞行为。该函数使用了某些假设条件，例如，贴图单元上端呈平面状，且法线为已知项。除此之外，假设条件还包括中心点的高度值以及尺寸为 16 的贴图单元。最后一个假设条件则是：其他表面均与坐标轴对齐。练习 23.1 还将要求读者编写相关函数，随着数据面的增加，该函数可对碰撞行为进行检测。上述方案的对应实现函数相对冗长，如下所示：

```
function box3DTileTopCollision(center, width, length,height, displacement,
                               tileCenter, tileHeight, tileNormal)
```

```
//width, length and height are the
//half-lengths of the sides of the box
//calculate tile top edges
set edge1 to crossProduct(tileNormal,(1,0,0)) //edge vector parallel to z-axis
set edge2 to crossProduct(tileNormal,(0,0,1)) //edge vector parallel to x-axis
multiply edge1 by 16/edge1[3]
multiply edge2 by 16/edge2[1]
//find vertices
set c to tileCenter+(0,tileHeight,0)
set v1 to c-edge1/2-edge2/2
set v2 to v1+edge1
set v3 to v2+edge2
set v4 to v1+edge2

set t to 2
//find collision time with base of box
if displacement[2]<0 then //box is going down
  set planec to center+(0,-height,0) //start height of base
  set vtop to the one of v1, v2,v3, v4 with the highest y-coordinate
  set t1 to (vtop[2]-planec[2])/displacement[2]
  //possible collision during time period
  if t1<1 and t1>=0 then
    //check for intersection within box base
    set p to vtop[2]-t1*displacement-planec
    //vector from box center to intersection point
    if abs(p[1])<width and abs(p[3])<length then
      set t to t1 //top vertex collides
    end if
  end if
  //NB: if there is a collision with this vertex,
  //it has to be the first collision
end if
if t=2 then //try other collisions with base
    if dotProduct(displacement, tileNormal)<0 then
    //find the vertex of the box that
    //would collide with the tile face
    set basevertex to planec
    if tileNormal[1]<0 then add (width,0,0) to basevertex
    otherwise add (-width,0,0) to basevertex
    if tileNormal[3]<0 then add (0,0,length) to basevertex
    otherwise add (0,0,-length) to basevertex
    //basevertex is the leading vertex
    //on the box with respect to the tile top

    set t2 to dotProduct(basevertex-c, tileNormal) / dotProduct(displacement,
                            tileNormal)

    //leading vertex intersects tile plane
    if t2<1 and t2>=0 then
      //check for intersection within tile
      set p to basevertex +t2*displacement-c
```

```
          if abs(p[1])<8 and abs(p[3])<8 then
            set t to t2 //leading vertex collides
          end if
        end if
      end if
      //NB: again, this collision will always
      //come before any other potential collision
    end if
    if t=2 then //try edge-to-edge collision
      set n1 to crossProduct((1,0,0), norm(displacement))
      //n1 is the normal of the plane swept
      //out by x-aligned edge of box
      set s1 to dotProduct(basevertex-vtop,n1)
      if s1>0 then
        set s1 to -s1
        set n1 to -n1
      end if
      //check for intersection with z-aligned axis of tile
      divide s1 by dotProduct(edge1,n1)
      if dotProduct(c-vtop,edge1)<0 then set s to -s1
      otherwise set s to s1

      if s>=0 and s<1 then
        set p1 to vtop+s1*edge1 //intersection point with edge
        set t3 to magnitude(p1-basevertex)/ magnitude(displacement)
        if t3<1 and t3>=0 then set t to t3
      end if
      || repeat for other edge pair
    end if
    if t<1 and t>=0 then return t
  end function
```

该函数采用了多项优化措施，并生成对齐盒体。例如，由于盒体边与轴向对齐，如图 22.3 所示，因而仅可能产生两个边-边碰撞。需要注意的是，标记为 vt 和 bv 的顶点与 box3DTileTop Collision()函数中的 vtop 和 basevertex 变量对应。

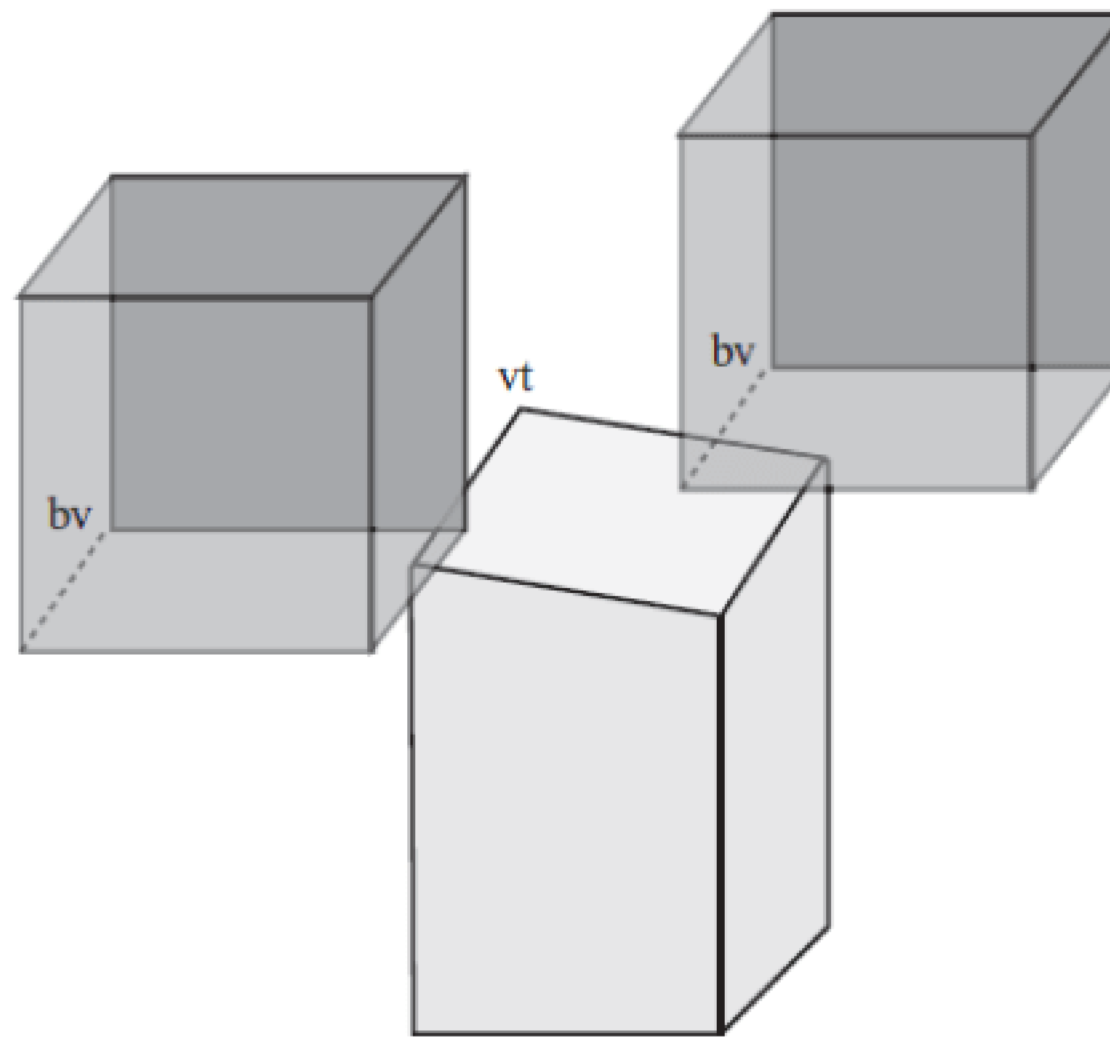


图 23.3　3D 贴图单元中可能的边-边碰撞

23.3.3 基于样条的贴图单元

作为 TGB 游戏的最后一个示例，本小节将讨论曲线路径。鉴于该方案的复杂性，这里仅对其进行简要讨论。尽管构建于 3D 场景中，但此类游戏仅限于单一路径，并以曲线形式穿越游戏场景世界，对应轨迹仅涉及 x 或 y 方向上的小范围区域。

实际上，读者对特定游戏的编程方式并无十分把握，但可适当猜测样条贴图单元的创建方法。方法之一是根据 3D 样条描述主路径，并于随后将其映射至简单的 2D 贴图单元中，如图 21.2 所示。

当加载基于样条贴图单元的游戏时，可将 2D 贴图转换至 3D 形式，但需要对贴图实现曲化操作并与样条匹配——此类方法可用于构造赛车类游戏。对此，可在平面或空间内（以支持太空类竞赛游戏）实现直线与曲线间的转化工作。

样条贴图单元使得全部碰撞检测可在简单的贴图单元空间内进行，并通过真实的 3D 空间完成显示功能。然而，为了实现健壮且自然的外观效果，该方案的工作量也会随之增加。

23.4 本 章 练 习

【练习 23.1】试完成 box3DTileTopCollision()函数，并对其进行适当扩展，以实现其他贴图单元数据面间的碰撞操作。在本章中，该函数之前的版本遗留了某些未完成的工作，读者可对其予以完善。随后，读者还可实现与其他数据面之间的碰撞计算。需要注意的是，面-面碰撞仅存在两种可能性，其实现过程较为直观，但应注意，顶端边与地面并非是平行关系。

23.5 本 章 小 结

与数学知识相比，本章着重阐述了相关方案的编程实现方式，此类话题在游戏设计中十分重要，理应得到足够的重视。本章也可作为第 24 章的预备知识，第 24 章将讨论基于网格的迷宫类游戏。

至此，读者应掌握如下内容：

- 如何构造并绘制 2D 贴图单元游戏。
- 通过复制预渲染图像或实时绘制方法实现卷轴操作。
- 如何控制相机对象进而生成相对自然的场景环境。
- 如何计算 2D 游戏或基于向量贴图单元的 3D 游戏的碰撞行为。
- 通过映射简单的 2D 描述，如何构建基于样条的游戏环境。

第 24 章　迷宫类游戏

本章包含如下内容：

- 概述。
- 迷宫分类。
- 创建迷宫。
- 在迷宫中漫游。

24.1　概　　述

迷宫类游戏可视为一种较为常见的游戏类型，多种游戏均可归类于这一类型，例如 Pac-Man 游戏。该游戏即使用了迷宫结构，并在大量的网格中实现游戏体验。在某些场合下，贴图单元游戏以及 3D 休闲类游戏也会使用到迷宫结构，并辅以简单的人工智能（AI）机制在迷宫中进行搜索。

24.2　迷 宫 分 类

在考察迷宫机构的工作方式之前，应理解该结构的正式含义及其划分方法。这里存在两种考察方式：方式一是查看其物理属性，包括基本网格的形状（矩形、三角形、圆形等）以及迷宫路径的走向；方式二则是考察迷宫结构的拓扑方式。严格地讲，拓扑是一类较为重要的数学领域，并将对象作为属性集进行研究，对应属性涉及变形效果（扭曲、翘曲以及弯曲等）。另外，拓扑结构可独立于迷宫的物理因素。当采用拓扑结构时，可通过链接路径的分支点集合检测迷宫结构。

24.2.1　图和连接性

从数学角度上讲，迷宫结构可视为一类图结构。然而，此处所使用的术语“图”具有与以往不同的含义。在当前上下文环境中，图由链接直线或边的多个点构成，如图 24.1 所示。其中，数据边构成的顶点称作节点。在图 24.1 中，左侧迷宫转换为右侧的图结构。在图中，各节点分别代表迷宫中的分叉、十字路口以及死端路径；各边则代表节点间的路径。除此之外，还存在两种特殊点且分别位于起始处和终止处。出于简单考量，此类数据点通过自身节点加以定义（S 和 F），尽管该点也可置于近分支点处或死端位置处。虽然迷宫结构并未强制要求初始点和终止点，

但该结构一般会包含这两个点。

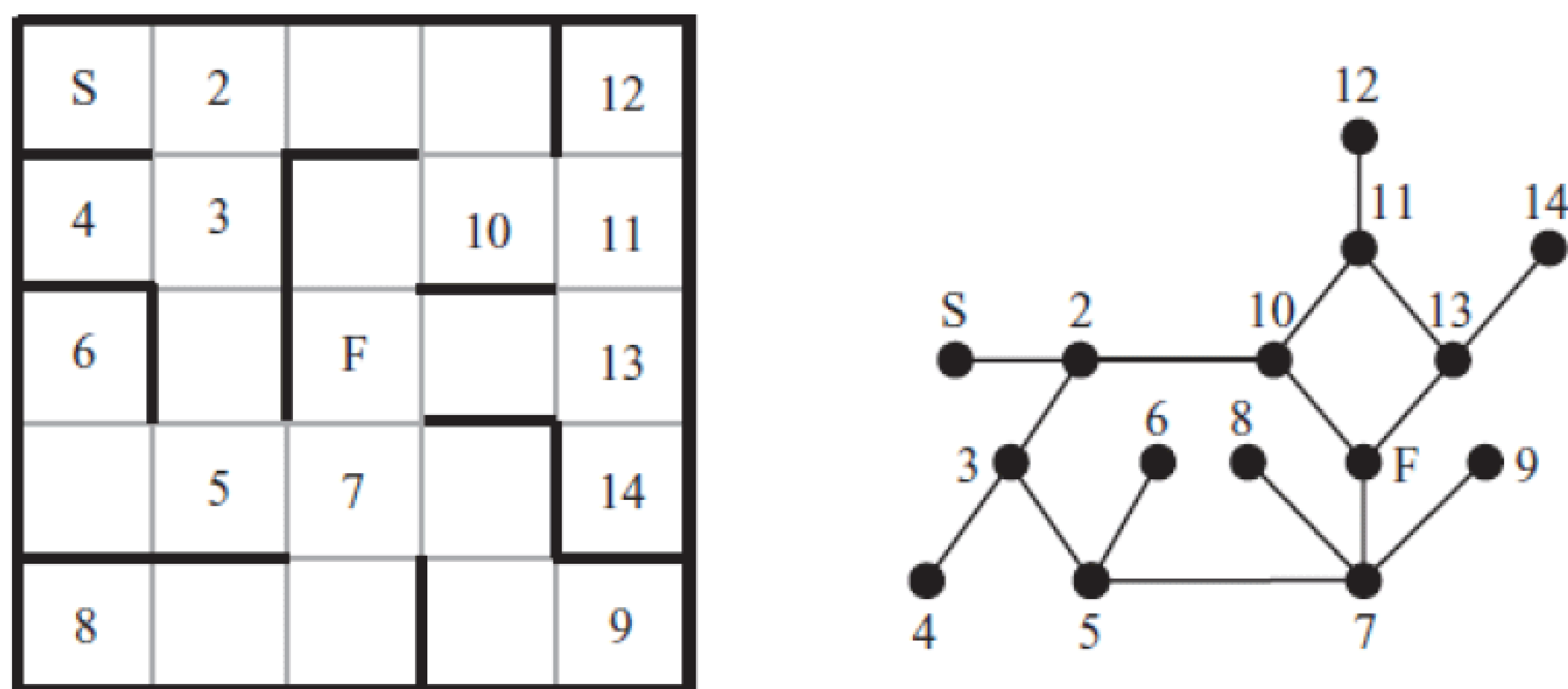


图 24.1　迷宫结构及其关联图

多数时候，人们多关注图的绘制方式，当从数学角度考察图结构时，关注点仅在于拓扑属性。也就是说，仅关注点之间的连通性。尽管如此，依然可为图中各边提供一个标记。该标记可显示一个表示为边长度的数值。同时，还可使用对应值（表示某种计算开销或距离）标记顶点（节点）。相应地，通过该方式标记的图称作网络。另外，若不希望定义正式的网络结构，则可采用边和节点标记索引图结构以供引用之需。

传统的图结构在任意顶点对之间最多包含一条边，且顶点间不可自行连通。然而，这一类限制条件并不适用于迷宫结构，因而某些时候需要考察所谓的伪图结构，迷宫即是其中的一例。通过移除干扰边，伪图可转换为真正的图结构，且不会影响到迷宫的主要属性，特别是点间的路径。由于不同的边选取方案可对图结构或网络结构产生影响，因而搜索最优路径时该结构应支持多连通。对此，通过插入附加顶点（在网络结构中通过长度为 0 的边进行连接），可将伪图转换为实图。

除此之外，还存在其他多种图结构的分类方式，并可应用于迷宫结构，如图 24.2 所示，相关内容可归纳为以下几点：

- 若各节点与其他节点之间存在路径，该图则具有连通性。图 24.2（a）即为连通图，而图 24.2（b）为连通图。
- 若任意两节点间仅存在一条路径，该图则称作树结构。任何包含 n 个节点和 $n-1$ 条边的连接图均可表示为一棵树，如图 24.2（c）所示。
- 若可在纸面上绘制图结构且不存在两条边彼此交叉，则该图称作平面图。针对图 24.2（a），图 24.2（d）显示了其绘制方式，并以此证明前者为平面图。另外，全部树形结构均为平面图。

迷宫（对应图为树形结构）定义为单连结构。抛开回溯操作不谈，此类迷宫包含一个正确的求解方案。若图连通但并非是树形结构，则对应迷宫结构称作多连通并包含环路。若图呈非连通状态，则存在某些无法到达的节点集。换言之，该图可划分为多个彼此不连通的连通子图。对于迷宫结构而言，这也意味着该结构可能无解（若起始节点和终止节点位于不同区域），或者迷宫的某一部分因无法到达而存在无关性。

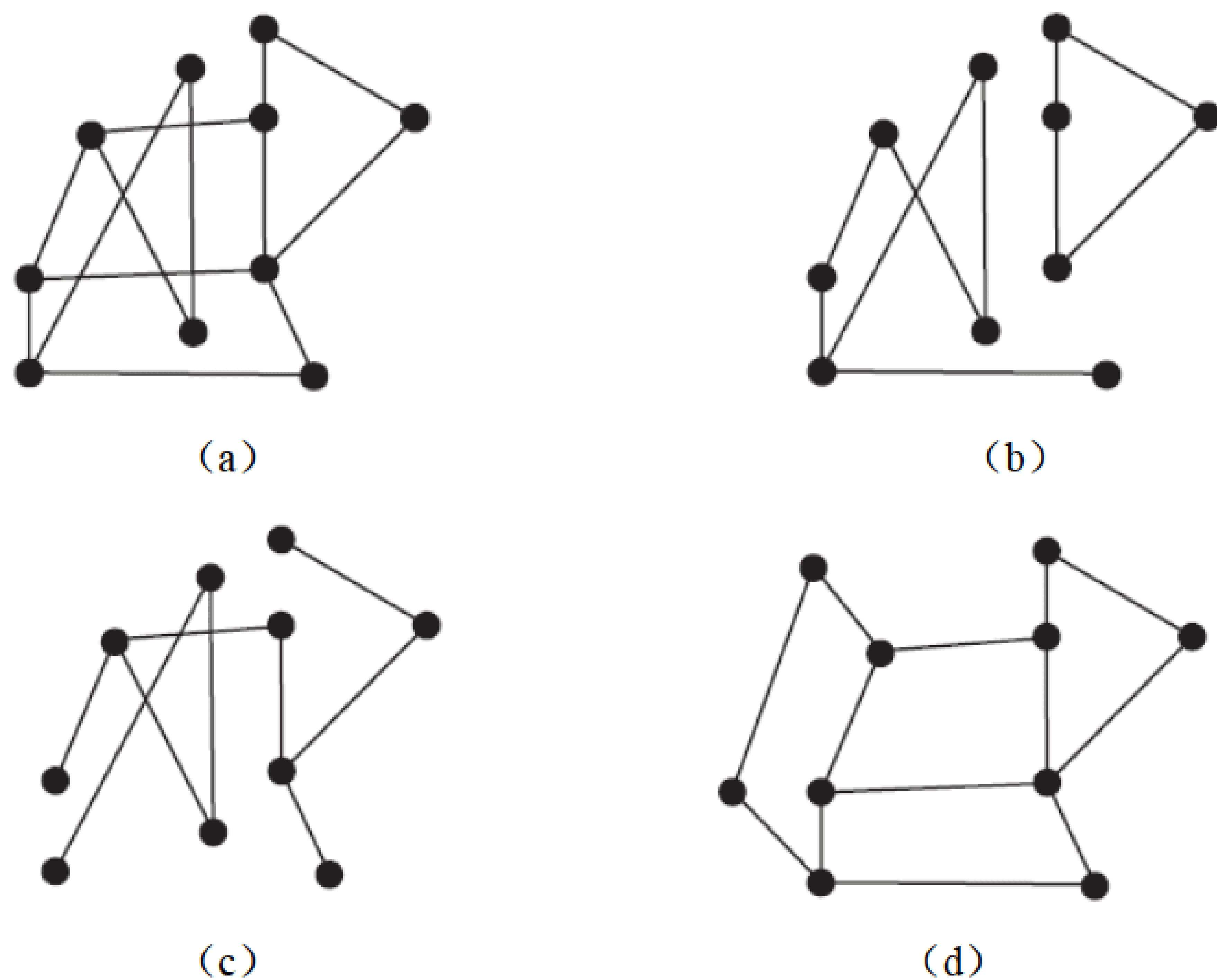


图 24.2　连通性和平面性示例

图论涉及广泛的内容和复杂的数学知识，其深入分析超出了本书的讨论范围。当在第 25、26 章考察搜索策略时，还将再次遇到图论问题。

24.2.2　迷宫转向

虽然迷宫的拓扑结构视为影响路径搜索的唯一因素，但实际上，其物理属性也同样重要。毕竟，真实的迷宫和供娱乐使用的篱笆式迷宫均不会采用复杂的图结构，而那些包含牛头怪物的经典迷宫甚至不包含分支点。相反，此类迷宫仅仅修建了通往中央位置的复杂路径，例如贴有大量马赛克装饰图案的迷宫建筑。因此，上述内容也体现了理论与现实之间的差异。

心理因素也会使得迷宫趋于复杂化，迷宫搜索游戏体现了视觉复杂度上的、人们急于到达目标的这一心理活动。若游戏为第一人称迷宫类游戏，则转向的记录难度对玩家的成功几率起到了重要作用。

心理因素的影响取决于迷宫的物理结构，因而有必要提供一份与迷宫物理特征相关的记录，下列内容显示了多个常见物理特征：

- 维度。迷宫结构是否包含一维、二维、三维或多维特征？其中，二维迷宫结果最为常见，但也可构造立方体形式的迷宫结构；或者，也可构造立方体序列并在其间通过入口予以连接。除此之外，还可尝试构建一类 2.5D 迷宫，并通过桥梁、隧道或瞬移连接 2D 迷宫的不同部分。对于更为复杂的迷宫，其自身还可在一段时间内产生变化，例如电影 Cube（拍摄于 1997 年）。
- 几何形状。2D 迷宫多采用平面加以表达，也可在不同方向实现环绕效果，例如 Pac-Man 迷宫，该迷宫在柱状表面进行绘制时，可自左至右环绕。此类行为彰显了迷宫的几何形状。另外，2.5D 迷宫的几何形状可描述为在 3D 空间内弯曲的 2D 迷宫。
- 基础网格。迷宫可根据正方形网格进行构造，如图 24.1 所示。另外，迷宫也可通过三

角形、六边形以及同心圆构建。同时，网格还可通过各种方式产生变形，例如透视转换。当然，迷宫结构也可不采用网格方式，此时，该迷宫表示为沿各分项延伸的多个墙面。

- 结构。迷宫结构可通过多种方式予以体现。一种方式是记录遇到分支点之前行进的距离；另一种方法是记录遇到通道（非分支）拐角之前行进的距离；第三种方案则是记录死端通道的比例，或者死端通道的平均长度。不同的迷宫生成方案将导致不同的迷宫结构。

在常见的迷宫特征中，其结构最为微妙。当采用基于特定结构的拓扑模式时，则可对迷宫结构进行深入考察。这里，多数拓扑结构均源自 Walter Pullen，他是 Daedalus 迷宫自由软件包的开发者（详细信息可参考附录 D）。

图 24.3 显示了两种与结构相关的不同迷宫，Pullen 将其称为径直度（river）迷宫。其中，左侧迷宫由较长的通道构成，并包含了少量的短程死通道，此类迷宫结构称作低径直度迷宫。相比较而言，右侧迷宫结构则称作高径直度迷宫，此类迷宫包含了较少的死端通道，但对应通道较长且相对复杂。需要注意的是，上述两种迷宫所涉及的最长通道彼此相同。通过计算迷宫死端路径的百分比，可向径直度中赋予某一数值。关于迷宫的求解难度，低径直度类型迷宫通常表现得尤为明显；由于高径直度类型迷宫仅包含较少的选择方案，因而易于处理。

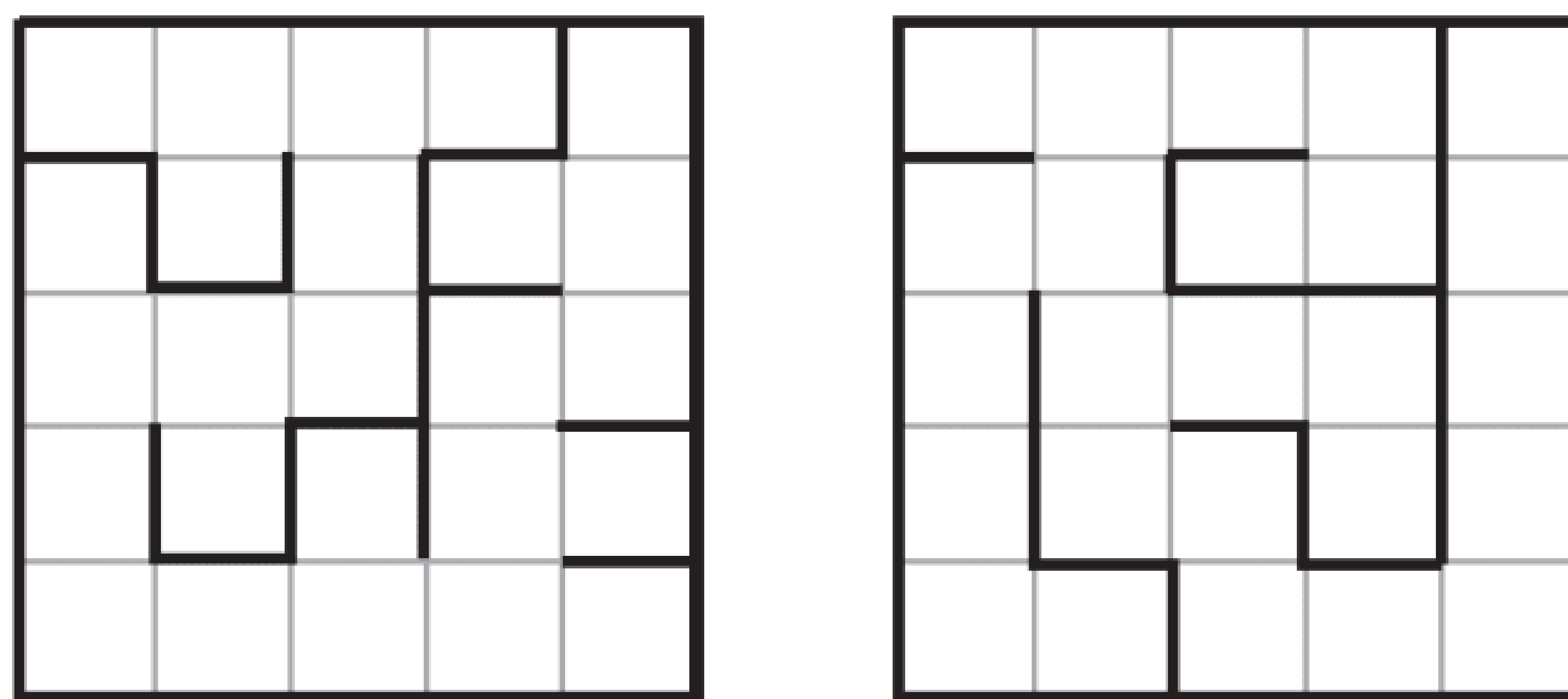


图 24.3　低径直度迷宫（左图）和高径直度迷宫（右图）

另一个与结构相关的术语是卷积，迷宫结构的卷积可通过基本树形终节点间路径长度的标准偏差进行计算。这里，标准偏差表示为一类常见的统计工具，并以此测算距平均数之间的距离。若一组数字 x_1，x_2，…，x_n 的平均数表示为 μ，则标准偏差 σ 表示为各数字距 μ 的均方差的根值。换而言之，平均数表示为方差的平方根（第 10 章曾对此有所讨论），标准偏差公式如下所示：

$$\sigma=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2}$$

统计学的深入讨论则超出了本章的学习范围，总而言之，标准偏差值越高，卷积值就越小。对此，较大的卷积值可通过蜘蛛网状的迷宫得到，进而包含大量的辐射于中心分支点的等长路径。此时，路径距离的标准偏差等于 0。相应地，另一种测算方案则是记录图中节点间的平均距离。尽管目标路径较长，但具有较高卷积的迷宫通常易于求解。

图 24.4 显示了两种迷宫，且不含相同的径直度以及不同的转折度。这里，转折度是指迷宫的转折程度，并以此表明以直线方式行进的距离。另外，转折行为还可在特定的方向上偏移，进而体现迷宫在不同方向上的长、短转折程度。例如，基于同心正方形的迷宫在不同点处包含不同的偏移转折值。

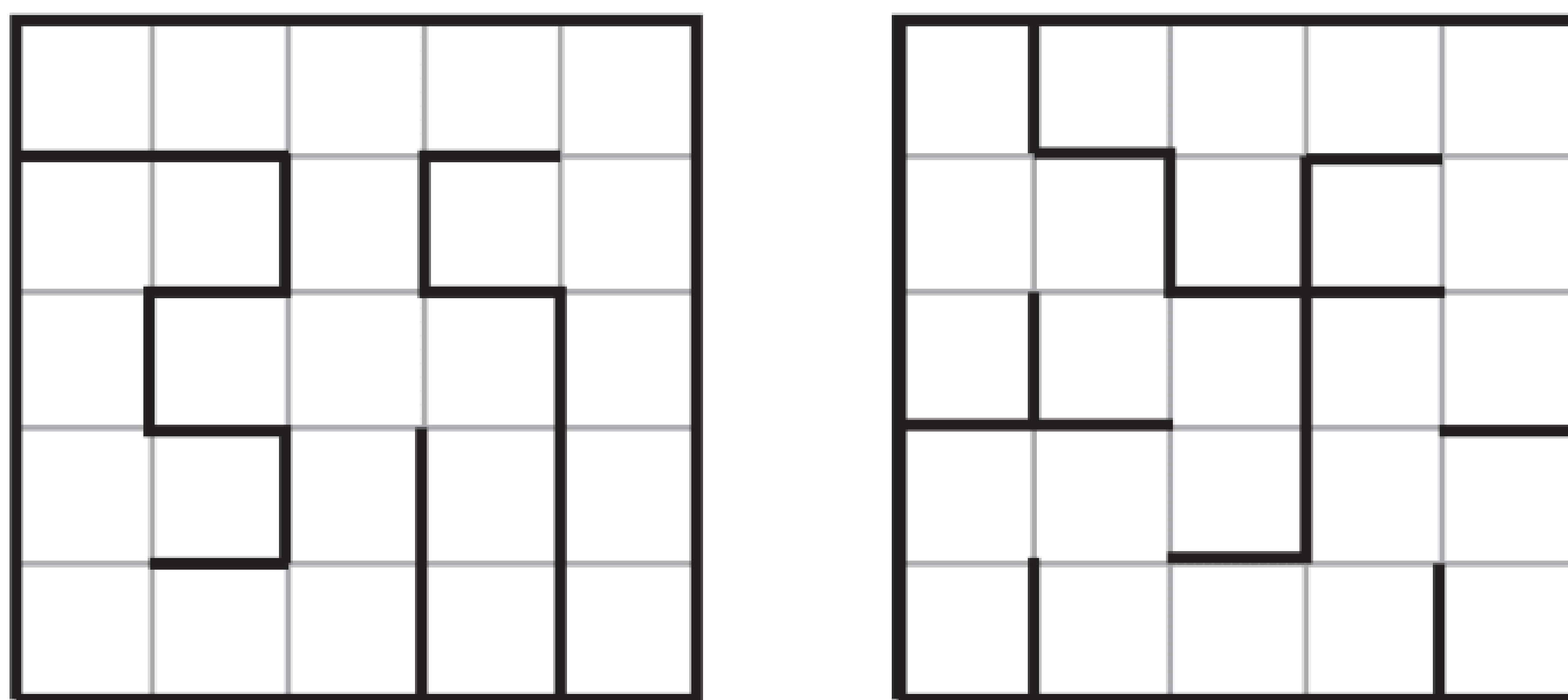


图 24.4　包含不同转折度的迷宫

24.3　生成迷宫

通过前述讨论，相信读者对迷宫结构存在大致的了解，下面将介绍迷宫在计算机设备上的处理方式，以及迷宫的随机生成技术。对此，可与基于网格的迷宫结构协同工作，此类迷宫结果通常可自动生成。大多数第一人称射击游戏场采用这一类迷宫，且与非网格迷宫具有显著的区别。非网格迷宫较少采用自动方式生成，相反，可通过关卡编辑器进行构造，并通过前述碰撞检测技术实现漫游操作。

24.3.1　处理迷宫数据

基于贴图单元的迷宫同样采用了网格技术，此类迷宫可定义为网格单元列表，并包含其间的多个墙面。例如，若正方形网格包含一个 $n×m$ 的单元阵列，且各单元均包含东、北向的墙面，为了确定网格单元(i, j)西向是否存在墙面，可查看网格单元$(i - 1, j)$是否包含东侧墙面，类似的技术也可用于南侧墙面的判断过程。回忆一下，在第 2 章中，曾采用 modulo()函数初始化此类网格。

正方形网格技术同样适用于其他网格，例如三角形网格，如图 24.5 所示。此类网格应视为两个包含不同属性且彼此交错的网格单元集合，或者一组可分割（或不可分割）的菱形集合。

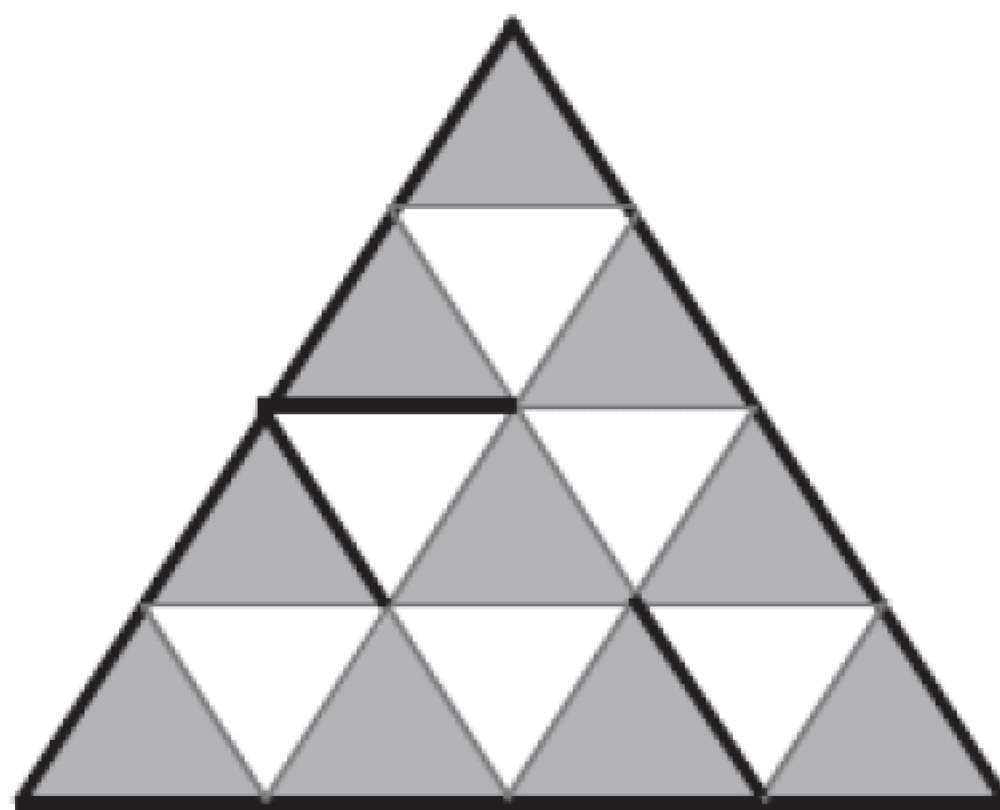


图 24.5　存储三角形网格中的迷宫数据

24.3.2　自动生成迷宫

这里存在多种标准算法可用于生成迷宫。总体而言，迷宫的生成方式可描述为：从边界处生成墙面，或者先期设置包含全部墙面（且基于贴图单元的）的迷宫结构，并不断移除某些墙面，直至完全实现迷宫。此处仅考察第一种方案，当迷宫按照前述内容进行构造时，该方案处理起来将更为方便。

最为简单的迷宫生成方案是递归回溯法，该算法维护路径内容，并在各步骤中于邻接单元与当前单元间进行查看，进而判断前者是否已被访问。若对应单元已被访问，则可尝试下一个单元；否则，若该单元未被访问，则可移除墙面并移至新单元中。若全部单元均被访问，则可沿路径回溯，直至获取一个未曾访问的单元。相应地，若返回至首个单元且未抵达任何邻接单元，则迷宫构造完毕，recursiveBacktrack()函数即采用了上述方案，如下所示：

```
function recursiveBacktrack(maze, startcell, endcell, path)
  if path is empty then add startcell to path
    set currentcell to the last cell in path
    set neighborList to the neighbors of currentcell in maze
    randomize neighborList
    repeat for each cell in neighborList
    if cell is not in path then
      set found to 1
      add cell to path
      remove wall between cell and currentCell in maze
      recursiveBacktrack(maze, startcell, endcell, path)
    end if
  end repeat
end function
```

Prim 算法可视为递归回溯法的变化版本，不同的是，该算法拾取路径单元的随机未访问邻接单元，其计算量也会随之增加——此处需要维护全部未访问邻接单元列表。然而，该算法的计算速度并不逊色，并可生成包含更多死端路径的迷宫结构。递归回溯算法以及 Prim 算法对于绘制软件而言均可提供填充操作，二者均可视为一类方便处理方案，进而生成包含不规则几何形状的迷宫，或者与任意基本网格协同工作。图 24.6 显示了上述两种算法生成的迷宫结构。

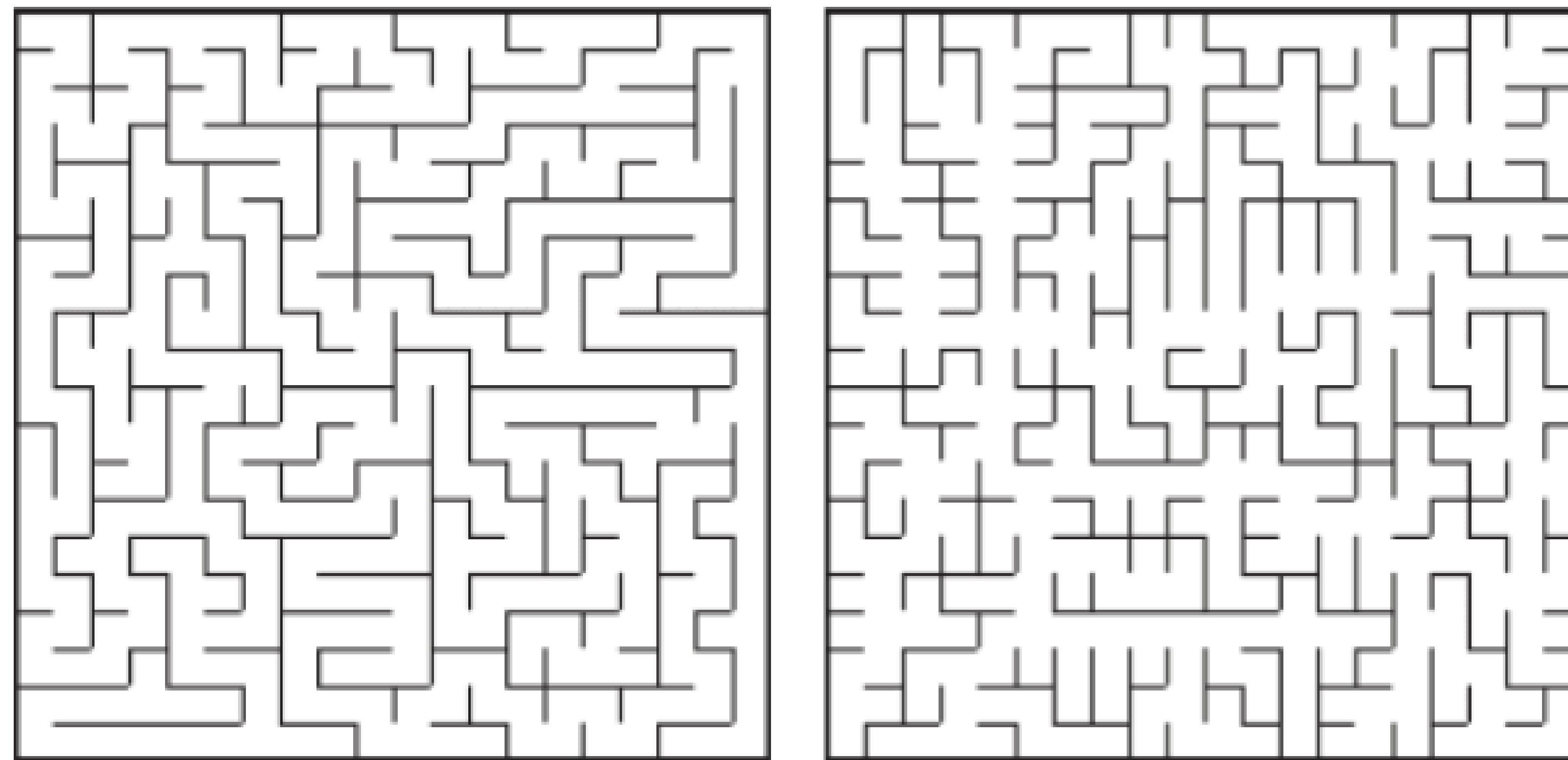


图 24.6　使用递归回溯法（左图）和 Prim 算法（右图）生成的迷宫结构

第三种算法称作 Kruskal 算法，且与前两种算法截然不同。该算法并不计算路径，且于开始阶段迷宫中的各个单元进行编号，并于随后随机选取墙面。若每一侧的单元包含不同的数字，则移除该墙面并针对各侧墙面使用相同数字再次进行编号。该操作过程以渐进方式构造迷宫中的连接区域，直至全部单元均编号为同一数字——此时，迷宫构造完毕。Prim 和 Kruskal 算法均为遗传算法示例，在第 26 章将对此再次进行讨论。kruskal()函数封装了上述行为，如下所示：

```
function kruskal(maze)
  set wallList to a list of all walls in maze
  randomize wallList
  set idList to an empty array
  repeat for i=1 to the number of cells in maze
    id maze[i] with i
    add [i] to idList
  end repeat
  repeat for wall in wallList
    set cell1 and cell2 to the neighbors of wall
    set id1 to cell1's id
    set id2 to cell2's id
    if id1 <> id2 then
      remove wall from maze
      repeat for each cell in idList[id2]
        add cell to idList[id1]
        id cell with id1
      end repeat
      set idList[id2] to an empty array
    end if
  end repeat
end function
```

此处，可对 kruskal()函数进行适当优化，即使如此，其优化结果在计算机设备上的运行效果并不优于 recursiveBacktrack()函数，但对于任意类型或形状的网格而言，该函数工作良好。

第 4 个函数源自欧拉算法，与前述内容相比，欧拉算法包含某些缺陷。例如，该算法仅适用于矩形迷宫，且在某一方向上存在偏移。尽管如此，欧拉算法依然具有快速和内存高效性等特征。当采用该方案时，一次将考察一行迷宫数据且与 Kruskal 算法较为类似。在某一特定行中，各个单元根据前一行中其他单元的连接方式定义数字 ID，在前一行与新行中任意连接集之间最多可得到一条路径，这避免了环路问题。总体而言，与前述方案相比，欧拉算法相对复杂。需要注意的是，在开始阶段，迷宫墙面均不存在，eller()函数封装了欧拉算法，如下所示：

```
function eller(maze, hfactor)
  set w to the length of a maze row
  set h to the number of rows

  //create first row
  set currentRow to ellerRow(maze, w, 1, 1, hfactor)
```

```
    //create body of maze
    repeat for j=2 to h
      set newRow to ellerRow(maze, w, j, w*(j-1)+1, hfactor)
      set testList to a list of elements from 1 to w
      randomize testList
        repeat for each i in testList
        set id1 to newRow[i]
        set id2 to currentRow[i]
        if id1 <> id2 then
          repeat for each element in newRow
            if element=id1 then set element to id2
          end repeat

          repeat for each element in currentRow
            if element=id1 then set element to id2
          end repeat
        otherwise
          add a wall between cell (i, j) and (i, j-1) in maze
        end if
      end repeat
      set currentRow to newRow
    end repeat
    //adjust final row to ensure span
    set idlist to an empty array
    repeat for i=1 to w
    if currentRow[i] is not in idList then
    add currentRow[i] to idList
    remove the wall between cell(i, h) and (i-1, h) in maze
    end if
    end repeat
  end function

  function ellerRow (maze, w, row, id, hfactor)
      set r to an empty array
      repeat for i=1 to w-1
        add id to r
        if random(1000)<hfactor then
          add 1 to id
          add a wall between cell (i, row) and (i+1, row) in maze
        end if
    end repeat
    add id to r
    return r
  end function
```

图 24.7 显示了 eller()函数实现的两个迷宫结构，并使用了不同的 hfactor 变量值。其中，将 1～1000 之间的数值（表示某种概率）赋予该变量。在图 24.7 中，这将影响到各方向上的行进因子，

且需要大约 50%的调整以生成可行的迷宫结构。

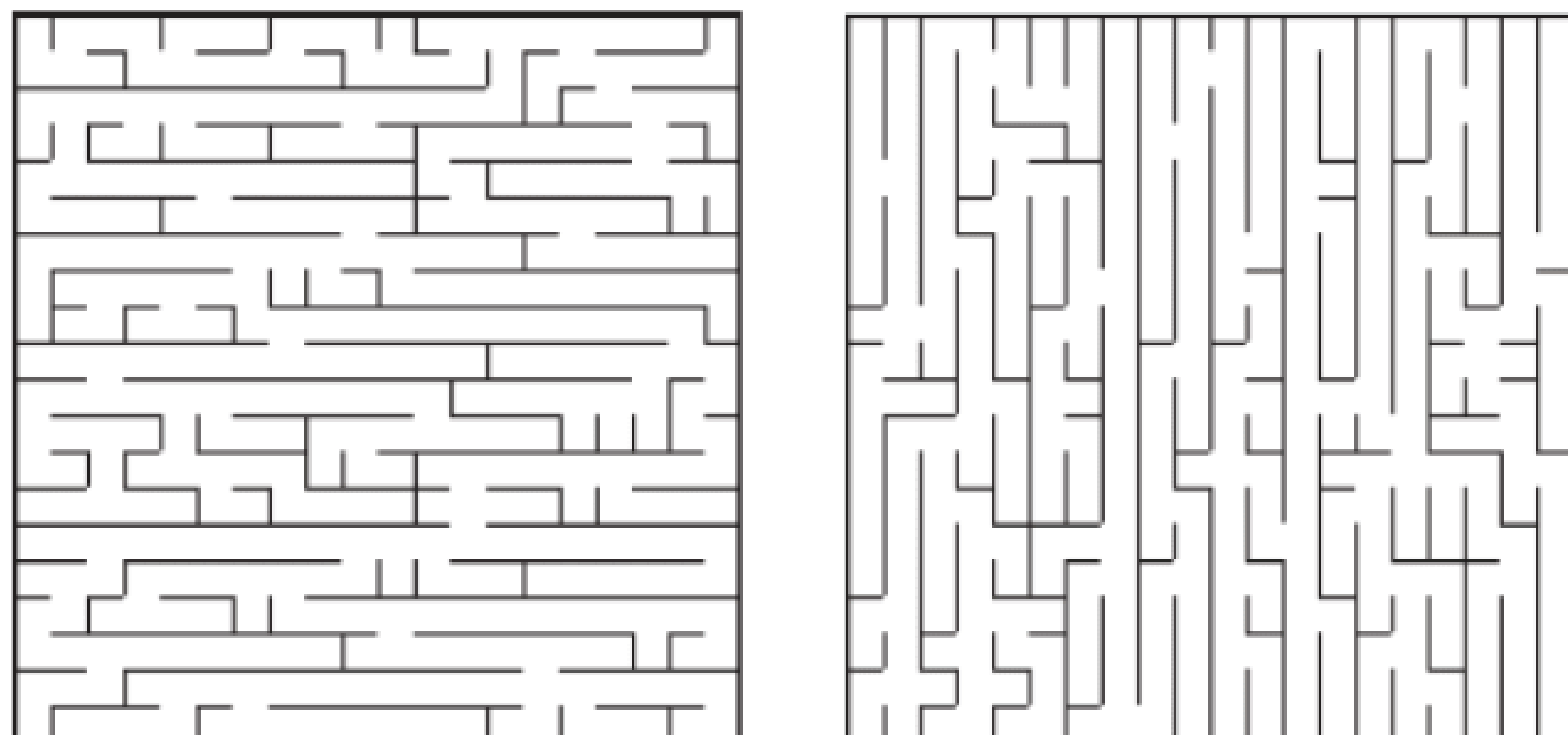


图 24.7　调整欧拉算法中水平墙面的随机因子（左：20%，右：80%）

欧拉算法可用于生成某一特定方向上任意大小尺寸的迷宫，这也是该算法的最大优点。除此之外，由于一次仅需存储一行迷宫数据，因而无须担心内存空间问题。同时，该算法同样适用于构造 3D 迷宫，且一次操作一个数据层（而非一行）。

24.3.3　多连通迷宫

需要指出的是，移除死端通道并不会降低迷宫结构的求解难度。实际上，问题通常会变得难于求解，但并非源自计算角度。从计算角度上看，无论迷宫结构是否包含环路，快速算法均以相同方式工作，尽管相对简单的算法（例如沿墙面行进这一类方案）可能会失效。对于大型迷宫结构，基于全路径存储的任何算法均会产生内存空间问题。

从心理学角度上看，环路的存在意味着人们更易于迷失方向。若将其作为迷宫结构的某一显著特征，这将增加操作的计算难度——环路使得随机多连通迷宫问题变得更加复杂。对于环路，对应过程将难以生成随机算法。从心理角度上讲，环路可生成更为有趣的迷宫结构。

生成多连通迷宫最为简单的方法是编写递归回溯算法的变化版本，具体而言，可在各路径一端向现有路径反向“开启”一个墙面。尽管该方法工作良好，但依然会生成包含环路。例如，最终结果可能是某一正方形中含有 4 个单元，或者中心位置包含单一墙面的 3×2 组合。

对此，一类计算密集型的高效替代方案可描述为，算法始于一个简单的连通迷宫，并于随后移除墙面。当移除墙面时，首先可通过随机方式选取若干墙面。针对各墙面，此处可计算该墙面两侧间的最短路径长度。对于较短的环路，可选取相应的中间路径，即采样中间的一条路径。该方案无法完全实现不包含死端路径的多连通迷宫（交错式迷宫），但其产生的迷宫结构通常较为有趣。multiplyConnected()函数即采用了这一方案生成迷宫，如下所示：

```
function multiplyConnected(maze, connections)
  //assume maze is already created and simply connected
  repeat with i=1 to connections
  set wallList to an empty array
  repeat for 11 walls in maze
    set s1, s2 to squares on either side of the wall
    set dist to path distance from s1 to s2
```

```
    add wall to wallList, sorted by dist
  end repeat
  remove the last wall in wallList

  end repeat
  return maze
end function
```

与简单的连通迷宫相比，多连通迷宫则更加复杂，与随机算法的生成结果相比，真实构筑的迷宫模式往往具有更好的效果。随机处理适用于迷宫重要区域（包含附加旋路和死端路径）间缝隙的填充操作。

24.3.4　更为复杂的迷宫结构

虽然网格式迷宫较为常见，但依然存在其他迷宫类型。从拓扑学角度上讲，网格式迷宫与其他子结构（或不存在任何结构）的迷宫之间基本等同，其差别主要体现于心理和计算方面。

圆形迷宫可视为一类较为常见的非网格式迷宫结构，如图 24.8 所示。依据拓扑学观点，该迷宫等价于图 24.1 所示迷宫，但前者的子结构有所变化。

对此，可通过多种方式构造图 24.8 中的迷宫，其中，一类高效的方法则是使用欧拉算法的变本。该算法并非逐行工作，而是从内向外以逐个圆的方式进行。与逐行操作相比，圆形操作方式则更为简单，二者间的唯一变化在于行两端处于环绕状态，对应处理过程等价于构造柱状迷宫。

待柱状迷宫创建完毕后，可将出口置于上下两端，并将全部内容映射为圆。如图 24.9 所示，底端出口位于迷宫中心位置，上方出口则位于外部。

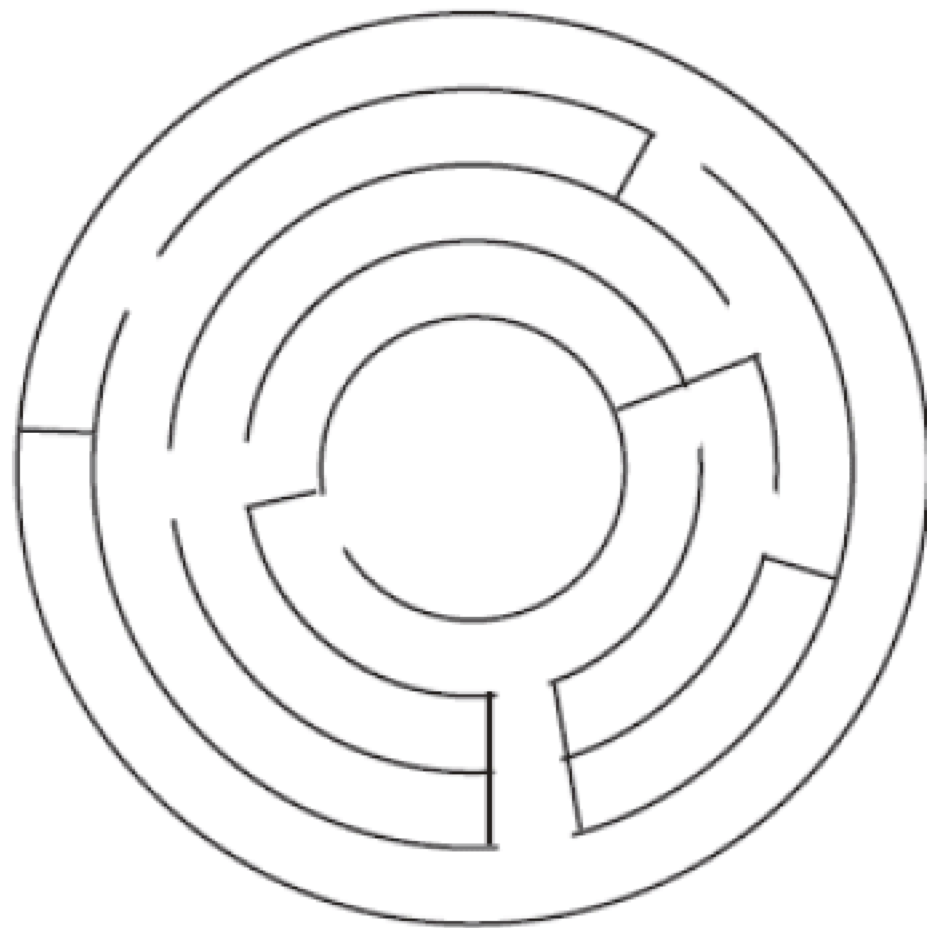


图 24.8　基于同心圆的迷宫

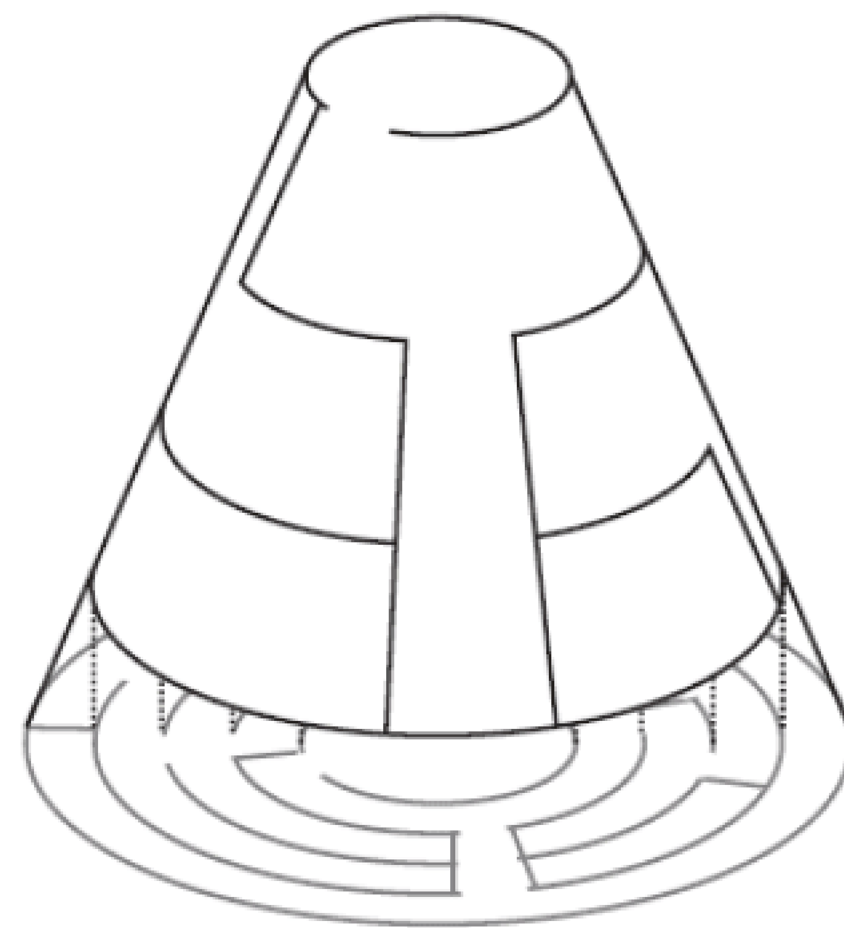


图 24.9　将柱状迷宫转换为圆形迷宫

图 24.9 所示的迷宫结构并未完全处于规则状态，与外圆墙面相比，内圆墙面则显得更为紧凑，即较小的空间内将包含更多的单元。当根据前述 hfactor 变量适当调整水平墙面密度时，该问题可得到有效的处理。另外，增加该变量值可在外侧行进时生成更多墙面。

关于迷宫的构造过程，其进一步的讨论则是如何在计算机上进行绘制。与直线网格绘制相比，圆形绘制方法通常较为困难。通常情况下，须通过不断调整方可获得具有较好外观的圆形迷宫。如图 24.10 所示的迷宫即为一类高效的自动生成方案，并通过计算机予以实现。

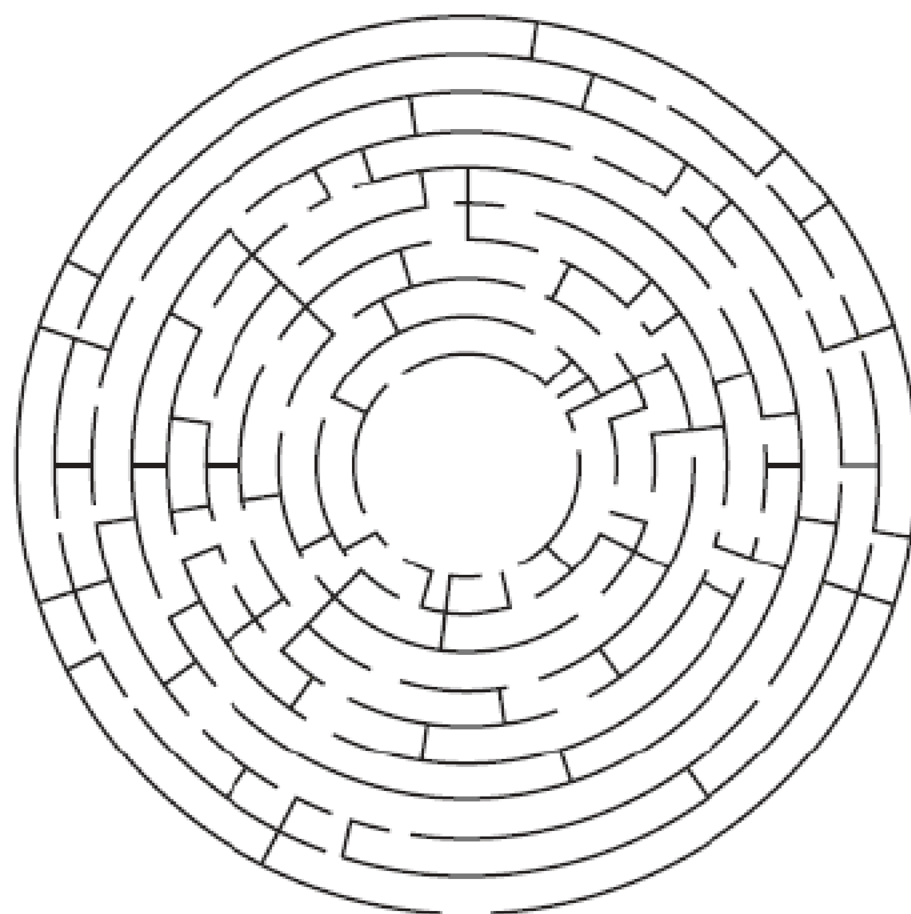


图 24.10　基于欧拉算法变化版本的圆形迷宫

24.4　迷 宫 漫 游

待迷宫结构构造完毕后，须对其执行漫游操作。当在迷宫中行进时，角色的尺寸变得尤为重要，也就是说，角色不可过大，以方便该角色在迷宫中的出入行为。当区分迷宫类游戏和贴图单元类游戏时，这可视为一个主要的判别条件。

单元间的行进可视为最为简单的迷宫移动方式，但这并不足够。角色须以平滑方式行进，其漫游方式也绝非视觉上的、单元间的运动，另外，漫游还应满足特定的运动路径，例如 A～B。毕竟，这体现了迷宫类游戏的主要目标。

24.4.1　碰撞检测和相机控制

迷宫漫游的主要技巧是将各个正方形视为独立的房间，多数时候，该模拟过程与四叉树十分类似，唯一差别是树形结构包含了地形部分。当游戏角色在房间内移动时，将会产生碰撞行为；当角色离开房间时，全部工作需要确定即将计入的房间，并检测途中是否存在墙面，该过程较为复杂。

代码的细节编写内容留予读者以作练习，但此处应注意一个问题，即墙面滑动（wallsliding）问题。此时，沿路径运动将导致角色进入墙面，而非停止或反弹。对此，可移除运动的法线部分，并保留切线部分内容。当角色抵达墙面时，将沿墙面移动，该结果相对自然且易于计算。

当采用第一人称 3D 视角且与墙面滑动方案协同工作时，相机应与墙面保持少许距离。否则，观察者仅会看到房间的部分视图。对此，较好的方法是将观察者视为一个球体，而非一点。

在第三人称 3D 视角中，由于须考察不同的碰撞集，因而计算过程变得更加复杂。例如，除了计算角色位置之外，还需进一步确定相机行进位置。总体而言，可将相机置于某一相对自然的静止点，且位于角色的后上方。如果可能，当角色移动时，相机应随之跟进，但这一方案并非总是可行，如图 24.11 所示。其中，位于 A 处的角色转向完毕，而之前 B 处的相机很可能位于 C，这将使其位于墙面内。

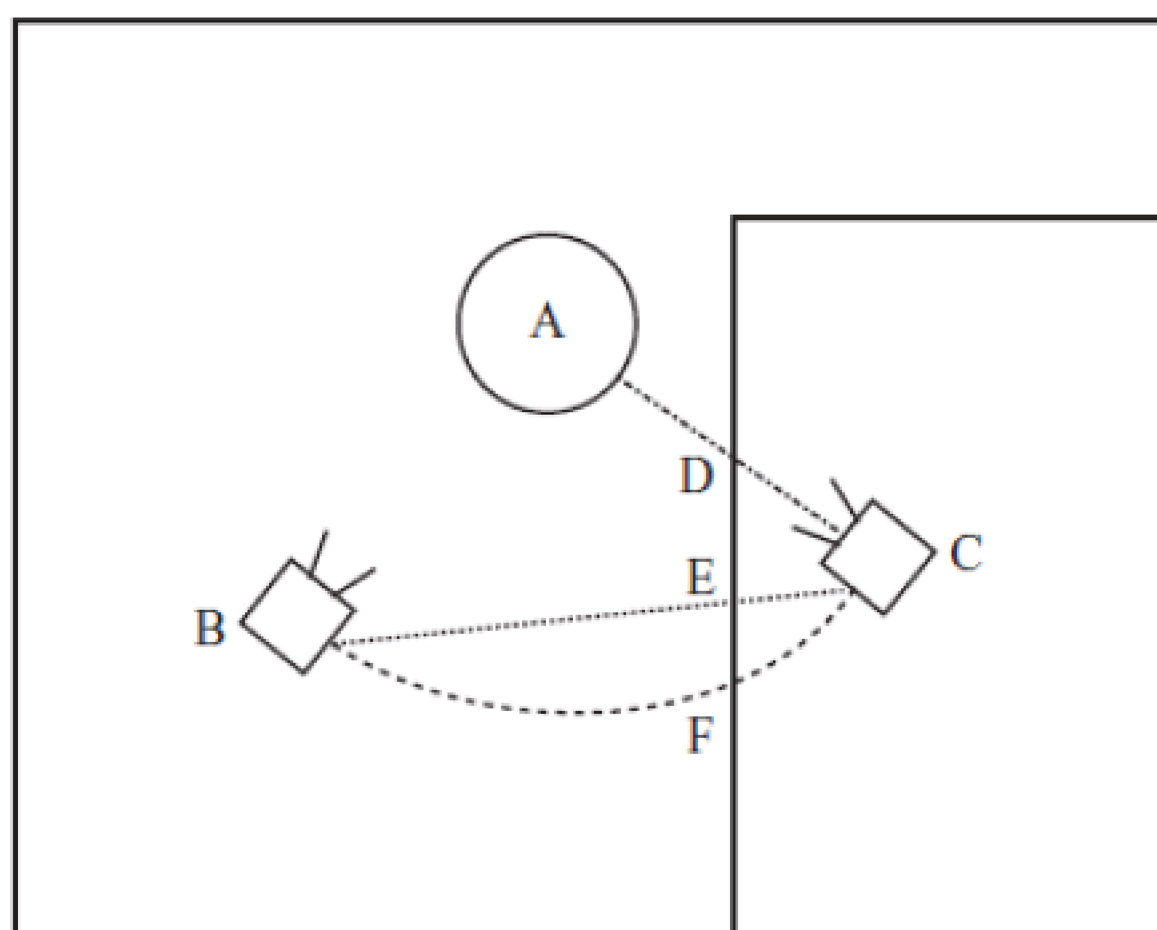


图 24.11　相机与墙面间的碰撞问题

当处理图 24.11 中的场景时，可将相机视为另一个对象，并包含自身的有难度路径，其碰撞处理与第一观察者所采用的方法相同，但仍需确定碰撞所涉及的具体数据。对于相机而言，较好的解决方案是使其停留于 B 点，即直线 AC 上的最近点。对应结果可能过于接近观察者，因而须适当提升相机位置，以使其在观察者处俯视场景。

除此之外，另外两个位置则是 E 和 F。对于点 E，相机先以 B～C 间的路径与墙面碰撞。另外，F 位于曲线路径上，且该路径不产生任何功效。上述两个替换位置面临同样的问题，即观察者无法看到角色所视的全部内容。

同时，针对如图 24.12 所示场景，若相机位置位于 C 处，则无须考虑其他位置点。此时，相机可直接跃至 C 处，而非 B～C 间的插值计算。

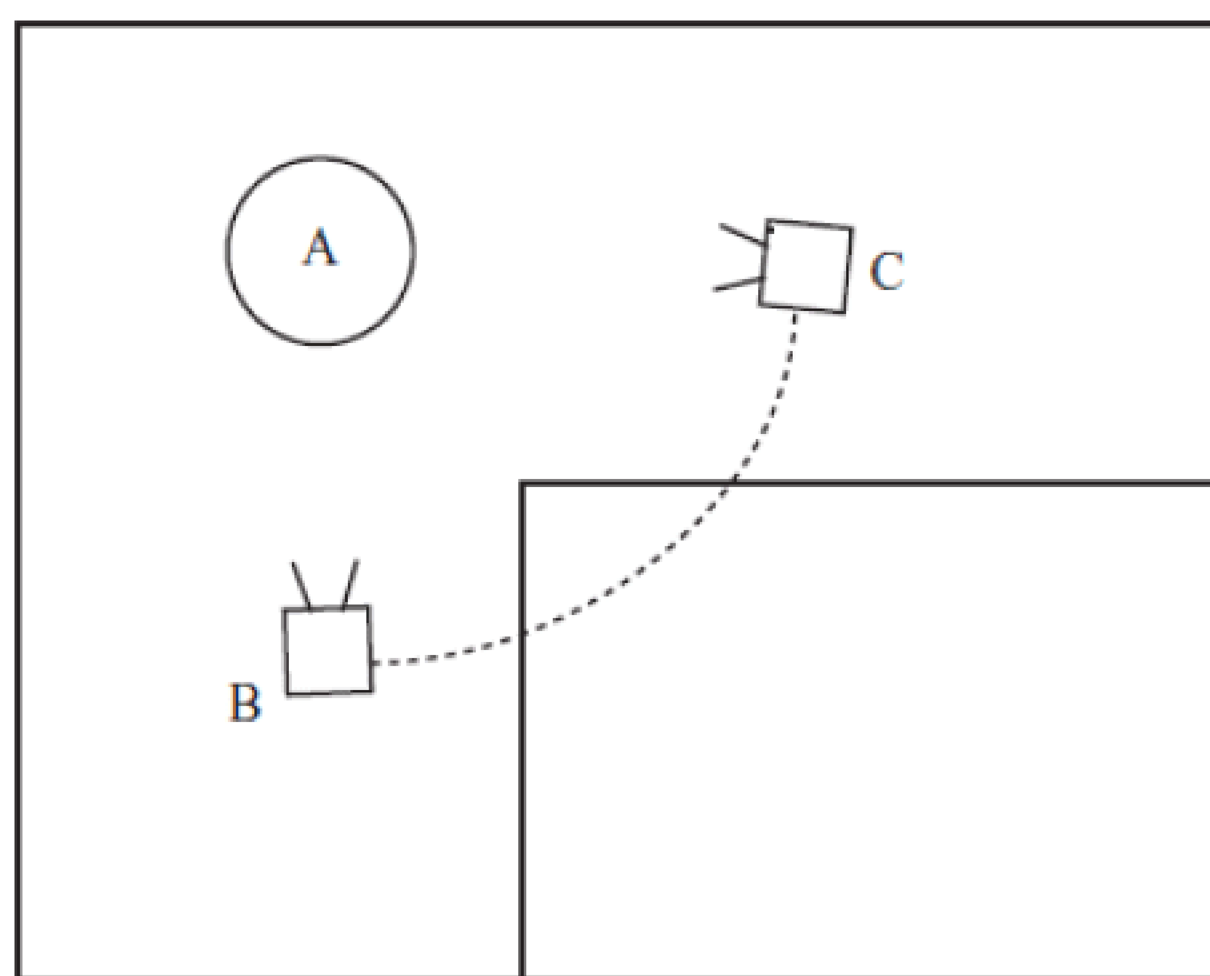


图 24.12　合力的相机运动

如前所述，上述相机控制方案与个人喜好相关，且需要谨慎处理。另外，用户应可处理系统中的某些异常情况。

24.4.2　视线

网格迷宫可相对容易地计算基于特定点的可见区域，这也是该类型迷宫的方便之处，同时，这一类问题也涉及视线的计算。视线计算常出现于 3D FPS 游戏中的可见性剔除操作，以及敌方

AI 的计算行为。例如 Pac-Man 游戏中的怪物或隐蔽类游戏中的哨兵。出于讨论目的，此处采用基于正方形的迷宫结构，相关技术同样适用于其他迷宫类型。关于视线算法，最简单的方法是考察相应的示意图，如图 24.13 所示。其中，对应角色位于符号“×”处的一点，该角色的全部可见区域采用灰色着色。需要注意的是，除了主行中的网格单元之外，还存在多个邻接单元也处于可见状态。

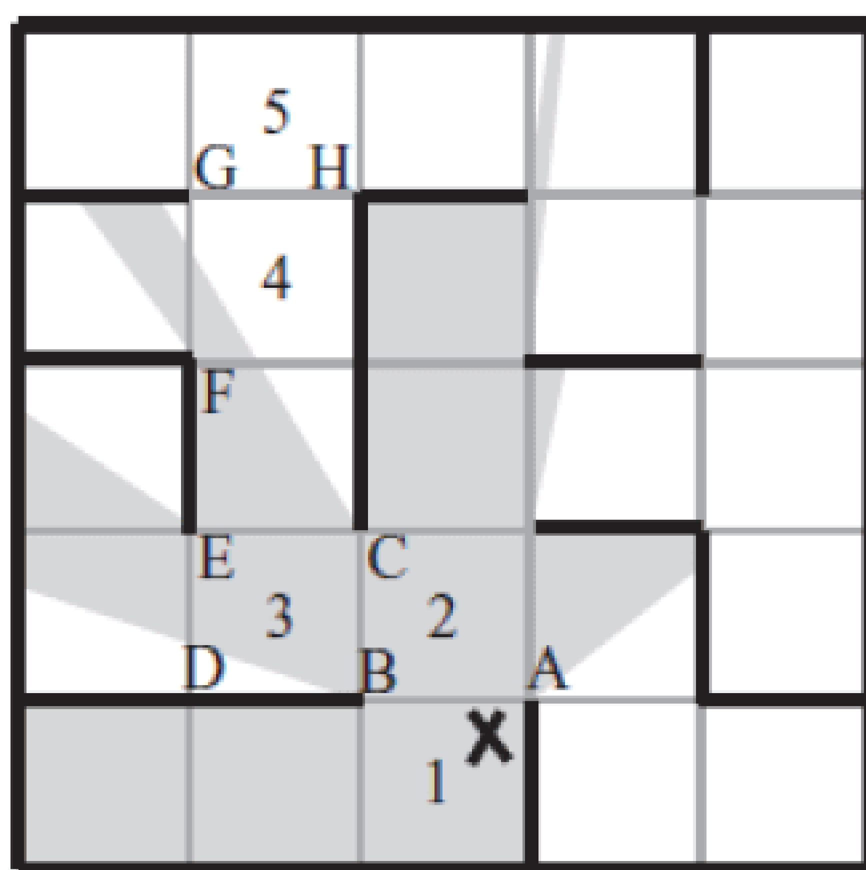


图 24.13　特定点的可见性计算

不难发现，即使在图 24.13 所示的网格中，其计算过程并不简单，但依然存在相关技巧可对此予以处理，尤其是单连通迷宫，即递归操作。当每次途径出入口时，即进入了一片单连通区域，该区域由出入口所定义的光线以及初始光线加以描述。也就是说，在每次经过出入口时，可将围绕观察者的 360°环视区域划分为较小的片元。

再次考察图 24.13，在点 X 处，存在 4 个光线可达的出入口。其中，两个出入口为闭合型，另外两个为开放型。对此，直线 AB 在 X 处对应最大角度(点 A 和 B 两点则在 X 处对应角 AXB)。当前，可查看标记为 2 的单元的邻接单元，且全部 3 个出入口均为开放型。此时，可对各出入口执行递归操作。此处，可进入出入口 BC 并到达单元 3，将光线进一步分为角 BXC。需要注意的是，由于顶点 D 位于“照明”角度外部，因而可对其予以忽略。然而，由于顶点 E 被照亮，某些光线依然会穿过出入口，特别是光线 BXE。鉴于已知 C 被照亮，这也意味着光线 EXC 整体穿越 CE。

通过上述方式可持续执行递归操作，直至到达无法逾越的障碍或诸如 GH 这一类出入口结构。这里，单元 4 部分可见，且被光线 FXC 照亮，但点 G 和 H 均位于该光线的外部。这也说明，不存在源自点 X 的光线可通过 GH；而且，鉴于迷宫的单连通性，可确定不存在光线可到达该点之前的任何单元。

visibleSquares()函数提供了视线算法的粗略实现方案，如下所示：

```
function visibleSquares(observerPoint, beamStartAngle,beamEndAngle, fromSquare,
                     thisSquare)
  if beamStartAngle is not defined,
    then set beamStartAngle to -pi;
      beamEndAngle to pi;
  if thisSquare is not defined then
    set thisSquare to observerPoint's square
```

```
    set ret to an empty array
    append thisSquare to ret

    repeat for each neighbor of thisSquare except fromSquare
      if there is a doorway from thisSquare to neighbor then
        set v1 to the first vertex of the doorway clockwise from the observer
        set v2 to the other vertex of the doorway
        //find angles of these vertices (in the range -pi, pi)
        set a1 to the angle of v1 with observerPoint
        set a2 to the angle of v2 with observerPoint
        if both a1 and a2 are between beamStartAngle and beamEndAngle then
        if a1<a2 then
        //vertices are on the same side of the angle range
        set start to max(a1, beamStartAngle)+0.02 — the increment is added to
                         rule out 'just visible' squares
        set finish to min(a2, beamEndAngle)-0.02
        //recurse over all visible squares
        set s to visibleSquares(maze, observerPoint,start, finish, thisSquare, n)
                               append all elements of s to ret otherwise
        //the doorway 'straddles' the angle range
        set start to min(a1, beamEndAngle)+0.02
        set finish to max(a2, beamStartAngle)-0.02
        //split the angle range into two parts
        //and recurse along both paths
        set s1 to visibleSquares(maze, observerPoint,finish, pi, thisSquare, n)
        set s2 to visibleSquares(maze, observerPoint, -pi,start, thisSquare, n)
        add each element of s1 and s2 to ret
      end if
    end repeat
    return ret
  end function
```

由于 visibleSquares()函数包含了某些数据值的重复计算，因而可考虑使用替代方案处理出入口跨越起始角和终止角这一问题。通常情况下，存在多种方法可实现该算法的加速计算，方法之一即是存储出入口的顶点值。类似地，也可采用第 22 章介绍的 Bresenham 算法，进而避免浮点数据和三角计算等需求。

visibleSquares()函数也适用于多连通迷宫结构，但有时需要在不同方向上考察同一正方形两次。实际上，当源自不同侧时，同一正方形的两部分内容可能均为可见。最终，递归操作常在多条不同光线中检测特定顶点的“照明”结果。对于单连通迷宫结构而言，其最大优点在于，若已知特定单元位于视线范围外，则途经该单元的全部内容均可在 3D 场景中予以剔除。然而，在多连通迷宫结构中，该方案较少采用。

24.4.3 迷宫的进程

当求解迷宫问题时，一般存在两种解决方案。方案一可描述为：当置身于迷宫某一特定点时，需要找到一条通往目标的路径，且无须了解当前迷宫结构；而方案二则是查看两点间的最短路径，

24.4.4 节将对此进行讨论，当前仅考察方案一，即迷宫的路径搜索，该算法的具体实现可参考练习 24.1，对应内容类似于迷宫结构的生成方案。

迷宫路径搜索始于一类经典方案，也就是说，每次遇到交叉路口时，均保持同一转向，这与单手贴至同一墙面行进十分类似。对于单连通迷宫，该方法切实可行；对于多连通迷宫，若沿外墙面从入口至出口行进，则该方案同样有效。然而，如果围绕中心位置存在环路，则无法获取一条路径至多连通迷宫的中心位置。类似地，同样也不会获取一条最短路径。尽管如此，但算法依然不先简洁性，且不会产生路径存储方面的压力。

与此相比，另一类方法则可生成快速的多连通迷宫求解方案，即相对简单的递归搜索，且等同于递归回溯迷宫生成方案。算法将跟随各条路径，在到达死端路径或遇到已访问的单元时，回溯并尝试另一条替换路径。算法甚至还可用于盲测试，即按照特定方向行进，直至遇到某一墙面。这里，墙面可视为长度为 0 的死端路径。

若已知通往目标的一条路径，可适当地改善递归回溯方案，即先期尝试将迷宫划分为较小区域的某条路径，例如，该路径可将迷宫结构一分为二，这将显著地降低算法的搜索时间。另外，从长远角度来看，算法还将提升路径搜索的有效性（即使无法到达目标）。

其他方法还包括使用标记系统。当采用该方法时，假设沿某一真实迷宫行进，并在途径墙面上予以标记。从计算角度上看，鉴于可通过某种方式“注释”网格单元数据，因而该方案可在内存中维护全部迷宫网格。除此之外，该算法等同于递归回溯法。

当实现标记方案时，可沿出入路径并在各连接处进行标记。例如，身后的单一标记可表示当前处于前进状态；当转向时，第二类标记则意味着当前通路行进完毕。为了保证各条路径均被访问，可优先进入一条未予标记的路径。针对递归方案，可将全部环路视为死端路径；当前向移动时，若途经已访问完毕的连接处（通过标记路径予以指示），则可于随后转向；当后退行进时，则会期望遇到对应标记，但仅应存在一条包含单一标记的通道，并据此顺利退出。

对于标记方案，当抵达目标后，还应得到一条标记路径并可回退至入口处，即一次性通路。从字面上理解，该算法可表示为进程（threading），其中，标记后的路径类似于一个线程，该线程构成了一个环路进而指示死端路径。待抵达目标后，拉动端点可绘制出全部进程的环路，并留下一条独立的路径可返回至初始点。

24.4.4　路径搜索和 A*算法

迷宫进程问题的另一面则是路径搜索，对此需要处理两个问题，即当前迷宫结构为已知内容，其次需要计算一条 A～B 之间的最短路径。

路径搜索问题包含多种解决方案。方案一是采用与递归回溯法相同的平均时间计算路径，但并非在确认之前搜索整条路径（深度优先搜索，参考第 26 章），而是同步搜索全部路径（宽度优先搜索）。宽度优先搜索了类似于 Prim 算法，并于前沿边界填充迷宫。在各个步骤中，可在各方向上行进一步。同时，各单元均记录与其关联的邻接单元。这也意味着待获取求解方案后，可沿当前路径回溯至开始点。根据定义，该起始点可能为最短路径。

除了宽度优先方法之外，还存在另一种高效的处理方案。该方案结合了深度优先和宽度优先搜索的优点，即 A*算法。aStar()函数实现了这一算法，其中，迷宫类型并不存在任何限制条件，

但应可通过某种方式描述两个节点间的距离。函数代码如下所示：

```
function aStar(maze, start, goal)
  set pathList to an empty array
  set d to distance from start to goal in maze
  append [d, 0, start] to pathList
  sort pathList on element 1 of paths
  //by estimated distance to goal
  repeat until break
    set path to pathList[1]
    //extend each path to all possible neighbors
    delete pathList[1]
    set currentSquare to the last square of path
    set previousSquare to the second-to-last square of path if any
    repeat for each neighbor of currentSquare except previousSquare
      set p to a copy of path
      //no loops
      if neighbor is not an element of p then
        append neighbor to p
        add 1 to p[2] //length of path
        set p[1] to p[2]+distance(neighbor, goal)
                      //distance underestimate
        append p to pathList //retaining the sort on element 1
      end if
    end repeat
    if pathList is empty then return "No path to goal"
    if pathList[1] ends with goal then return pathList[1]
  end repeat
end function
```

A*算法的重要特征可描述为：该算法根据与目标位置间的测算距离排序现有路径。也就是说，算法优先搜索近目标路径，这与第 10 章讨论的 collision halo 碰撞检测方案有几分类似。

作为一类较好的迷宫搜索算法，在经过精心设计的迷宫中（路径经过特别设计并可抵达目标），A*算法的计算速度并非最佳。然而，针对随机迷宫结构，该算法则具有快速的特征，并适用于计算最优局部路径，即近目标路径，此类路径很可能是角色靠近目标时所采用的路径。

对于 A*算法的细节内容，该算法可始于目标正方形。在各个阶段中，可选取当前最佳路径，并将其扩展至全部可能的邻接节点。若存在一条环路，则可忽略该路径，否则可继续测算一条新路径，并将其添加至路径表中，以进行排序操作。

在图 24.14 中，标记为虚线的路径为 A*算法首先采用的路径，在开始阶段，路径尚呈现为规则的对齐状态。在图中转向过程中，路径长度增加且距目标间的距离也处于增长状态。图中，算法最后一次测算的增加结果超出了最初的锯齿状路径。最终，算法将切换到一条新路径，并快速抵达目标，即使沿原路径存在一条更为合理的路线。

最后，执行 A*算法时，尽管沿某一路径到达目标，但由于另一条具有最大测算值，该目标可能会置于路径表中的其他位置。

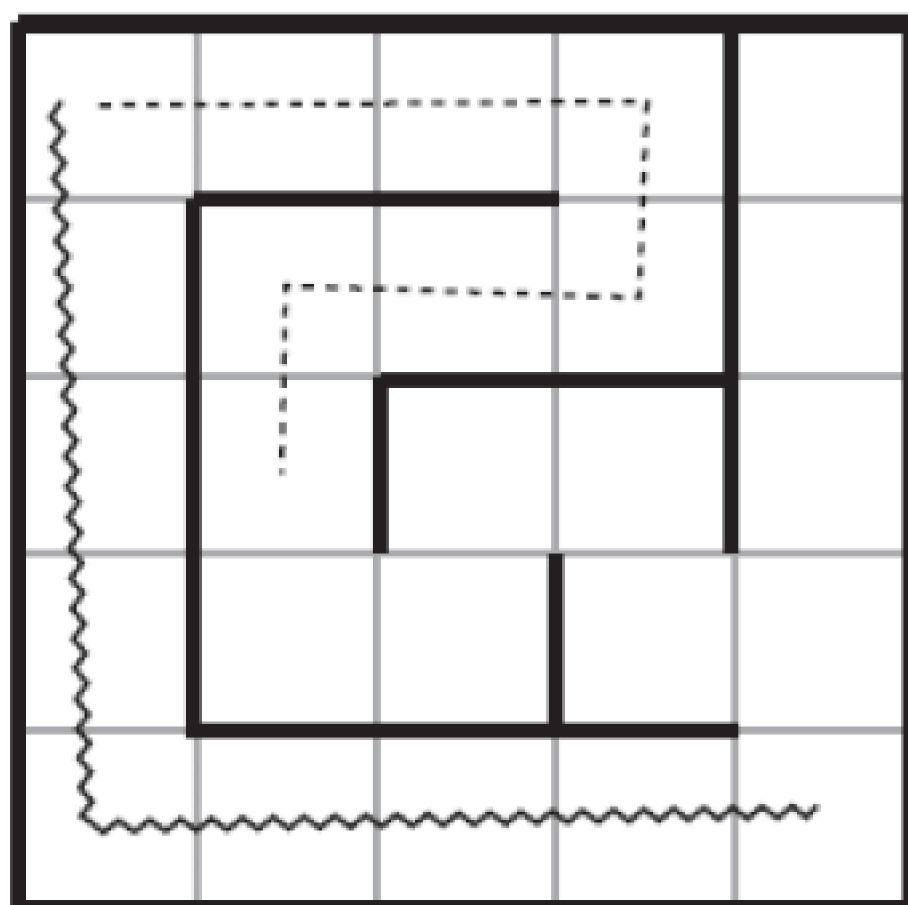


图 24.14　A*算法

24.5　本 章 练 习

【练习 24.1】试编写应用程序，并根据墙面跟踪法（wall-following）或递归回溯法漫游迷宫，读者可借鉴本章所讨论的某些算法。

24.6　本 章 小 结

本章深入讨论了与路径相关的细节内容，其中包括迷宫结构的生成和漫游方式，并先期介绍了搜索、AI 以及游戏理论，第 25、26 章将再次对其加以分析。特别地，第 25 章将充分利用本章介绍的图论知识。

至此，读者应掌握如下内容：

- 根据物理或拓扑属性对迷宫结构予以分类。
- 图论的基本知识，例如节点、边、树、网络以及连通属性。
- 如何在计算机设备上存储网格迷宫结构的细节内容，并据此实现迷宫漫游和相机控制操作。
- 如何通过特定算法构造迷宫结构，例如 Prim 算法、Kruskal 算法以及欧拉算法。
- 迷宫的漫游算法，例如 A*算法。

第 25 章　博弈论和人工智能

本章包含如下内容：

- 概述。
- 博弈论简介。
- 战术型 AI。
- 自顶向下型 AI。
- 自底向上型 AI。

25.1　概　　述

本章着重讨论人工智能（AI）以及该技术在大量游戏中的应用方式。这里，AI 是指机器的“思考”方法，该领域的研究同计算学自身一样历时已久。阿兰·图灵（1912—1954）是这一领域内的重要人物，作为一名数学家和计算机科学家，他曾发表了一篇重要的学术论文。当前，仅通过有限的方式实现了人工智能模拟，AI 依然是一项重要的认知科学领域，科研工作者依然在不遗余力地开发不同的思考模型。

AI 也是游戏设计中的一项基本内容，但游戏设计者较少负责设计具有真正思考功能的程序。科学家负责开发称作强人工智能的感知技术，从而使得机器具有人类的思维功能。相比较而言，游戏设计人员仅涉及弱人工智能，以使游戏可模拟思考行为，或以真实方式与输入设备进行反馈。尽管如此，两个不同领域之间依然存在相同点，即构建规则集进而生成灵活的行为以及学习功能。

25.2　博弈论简介

除了图灵之外，游戏和计算机史上的其他著名人物还包括 John von Neumann（1903—1957）、Oskar Morgenstern（1902—1977）以及 ÉmileBorel（1871—1956），且均被认为是博弈论方面的先驱人物。博弈论是一门与方案选取相关的学科，博弈论制定相关的游戏环境，玩家根据他人之思考、行为方式制定相关策略。1994 年，John F. Nash 等人对该问题实现了重大突破，并凭借博弈论获得了诺贝尔奖。2001 年，他富有传奇色彩的一生被拍成了一部电影，即《美丽心灵》。

25.2.1　零和游戏

零和游戏可视为博弈论中基本的游戏类型，其中，多名玩家根据各自策略执行相应的游戏体

验。据此，赢家获得输家所损失的利益，其目标可表示为最大化收益（或最小化损失）。这里，术语“零和”表示游戏中全部资金数量为常数，当一方收益后，另一方必定受损。

出于简单考量，本章所讨论的游戏均为双玩家游戏，相关技术也适用于多玩家游戏。图 25.1 显示了一款相对简单的双人零和游戏，该游戏类似于常见的石头、剪刀、布游戏。这里，假设游戏包含两名玩家 Andy 和 Beth，且二者同时伸出手掌，可供选择方案包括石头（R）、布（P）和剪刀（S）。若两名玩家手形相同，则二者平局且彼此并无损耗。否则，二者间的胜负关系可表示为 R > S > P > R，图 25.1 中显示了对应的胜负关系。其中，3 行数据确定了 Beth 可能的手形，3 列则表示为 Andy 可能采取的行为。另外，表中单元格所显数值表示为玩家的收益。

图 25.2 显示了另一个示例，该游戏称作 Undercut，并出现于数学家 Douglas Hofstadter 所撰写的《*MetamagicalThemas*: *Questing for the Essence of Mind and Pattern*》一书中（该书于 1985 年由 Basic Books 出版社出版）。其中，Andy 和 Beth 选择 1～5 之间的数字，选取较大数字的玩家所赢得的分值为两个数字之差；若两个数字相差 1，则另一名玩家赢得的分值为两个数值之和。

		Andy		
		R	P	S
Beth	R	0	-1	1
	P	1	0	-1
	S	-1	1	0

图 25.1　双玩家零和游戏

		Andy				
		1	2	3	4	5
Beth	1	0	3	-2	-3	-4
	2	-3	0	5	-2	-3
	3	2	-5	0	7	-2
	4	3	2	-7	0	9
	5	4	3	2	-9	0

图 25.2　Undercut 游戏

需要注意的是，并非全部游戏均为零和游戏，图 25.3 显示了博弈论中的一个著名的游戏，即囚徒问题。此时，两名参与者并非处于竞争关系。相反，二者将得到源自第三方的最大化收益。也就是说，两名囚徒将最小化监狱中的服刑时间。在图 25.3 中，Andy 和 Beth 损耗数据以向量形式加以表示。

		Andy	
		C	D
Beth	C	(3,3)	(5,0)
	D	(0,5)	(1,1)

图 25.3　囚徒问题

【提示】囚徒问题于 1950 年首先由 Merrill Flood 和 MelvinDresher 提出，并可通过多种方式进行描述，其中的一个版本可表述为：两名嫌疑人由于涉嫌犯罪而被拘捕，并于两个独立房间内进行审讯。若二人招供，则满足相关条件并规定了可能的处罚结果。首先，若嫌疑人甲供认，而另一嫌疑犯未供认，则甲将免于刑责，则嫌疑犯乙将获得全部惩罚；其次，若二人供认不讳，则均会受到惩罚，但刑期将会适当缩短；最后，若二者均不招供，则存在一定的免罪机会；同时，若证据确凿，则二人均会受到最大限度的惩罚。这里，两名嫌疑人的目标均是受到最轻程度的判罚，因而矛盾不可避免：各方均会思考对方的审讯结果。从实际情况来看，二人均供认不讳则是一种获益行为。事实证明，当人们体验此类游戏时，通常会选择拒不认罪。

25.2.2 求解游戏

从理论上讲，由于各玩家均会制定较好的策略并企图获得最大化收益，因而任意零和游戏均可求解。例如，图 25.4 显示了 Rube 游戏的矩阵数据，Rube 游戏和囚徒问题之间的差别在于：在 Rube 游戏中，游戏参与者可彼此欺骗。

对于图 25.4 所示的 Rube 游戏，假设 Beth 在打牌时作弊并支付了 100 美元的最大现金额度，Andy 则是这场骗局中的受害者。此时，Andy 和 Beth 选择 J、Q、K 中的一张牌。图中显示了对应的支付矩阵，不难发现，Beth 的最佳策略是选择 K，而 Andy 的最优方案则是选择 J，即 Andy 每次向 Beth 支付 1 美元。

		Andy			
		J	Q	K	Min
Beth	J	-100	2	-100	-100
	Q	-50	-100	2	-100
	K	1	2	5	1
	Max	1	2	8	

图 25.4　Rube 游戏

矩阵包含了 Min 值和 Max 值，对于某张特定的扑克牌，这分别表示了 Andy 向 Beth 支付的最大额度以及最小额度值（或者，Beth 向 Andy 支付的最大额度）。在上述两种情形中，此类值表示针对某张扑克牌的最坏方案。

每位玩家的首选方案即是选择对应扑克牌，并可最小化向另一玩家所支付的最大额度，即最小-最大策略。据此，Andy 应选择 J，Beth 则需要选择 K。需要注意的是，在支付矩阵中，Andy 和 Beth 均包含了最小-最大值 1，对于各玩家而言，这不失为一种稳健的策略。若 Andy 选择 J，则 Beth 的最佳选择方案是 K，其他牌面将导致较差的结果。另外，Beth 将对表中的负值执行最小-最大操作。

如果 Beth 选择 K，则 Andy 除了选择 J 外别无他法。随后可知，若各玩家均采用当前策略，则另一名玩家也需要以该策略予以应对，通过该方式实现的组合称作纳什均衡。一种基于该方案的理论表明，各有限、双人零和游戏须包含一个或无限多个基于该属性的策略组合，许多二人以上的多玩家游戏也支持这一理论。

对于 Rube 游戏，纳什均衡理论各次可选取单一行动，且 1 美元的支付费用称作游戏值，由于各玩家的最小-最大值保持一致，因而该游戏被严格定义。当采用不同方式描述时，支付矩阵中的值 1 表示列中的最小值和行中的最大值，该结果称作矩阵的鞍点。

然而，石头-剪子-布游戏则无法通过相同方式予以确定，该游戏的支付矩阵并不存在鞍点。实际上，各玩家甚至不存在独立的最小-最大值。鞍点的存在意味着，不存在单一策略适用于各玩家。对于 Andy 所制定的任何选择方案，Beth 均可制定对应的胜算方法。特别地，二人须在 1/3 时间值内准确地选取随机方案。该游戏的有趣之处在于，人类并不擅长随机选取数字。据此，一名玩家可尝试观察另一名玩家的行为模式并可抢先执行。

若游戏不包含鞍点，则需要制定相应的混合策略，并需要得到概率论的支持。这里，概率是

指位于 0～1 之间的数字，表明事件集中某一事件出现的可能性。相应地，公平硬币（fair coin）是指落地时硬币正反面的概率均等，即正面的概率为 0.5。对于六面骰子，标号为 6 的出现概率为 0.167。总体而言，离散事件的出现概率通过下式加以定义：

$$P(\text{事件})=\text{事件产生方式数量}/\text{全部可能事件数量}$$

针对某一骰子，由于存在两种方式可产生小于 3 的数字，且包含全部 6 种可能事件，因而掷出小于 3 的数字的几率为 $\frac{2}{6}=\left(\frac{1}{3}\right)$。

假设每掷出一次骰子，须向 Mary 交付 3 美元，且 Mary 向玩家支付 1 美元/点，对此可计算期望收益额度。若玩家计划长时间体验这一游戏，则对应额度表示为每场游戏所赢的平局值，即加权平均值，如下所示：

$$\text{期望支付额度}=\sum_{\text{事件}}P(\text{事件})\times\text{事件支付额度}$$

针对 Mary 的出价，期望支付额度为下列求和结果：

$$\frac{1}{6}\times1+\frac{1}{6}\times2+\frac{1}{6}\times3+\frac{1}{6}\times4+\frac{1}{6}\times5+\frac{1}{6}\times6=\frac{21}{6}=3.5$$

因此，除去初始支付额度 3 美元，期望收益为 0.5 美元，因而该游戏尚可参与。

游戏的最优策略可描述为各选取方案的概率集，无论其他玩家如何选取，期望概率保持不变。对于包含鞍点的矩阵而言，最优策略则相对简单，即选取概率为 1 的选项。对于其他情况，则需要通过代数方法进一步确定其概率。在 Undercut 游戏中，图 25.2 显示了 Beth 和 Andy 的数据矩阵。鉴于 Undercut 矩阵为对称矩阵，因而可知期望支付额度须为 0。若选择了概率为 p_i 的数值 i，对于 Andy 且基于特定策略，这将生成 5 个方程，如下所示：

$$\begin{pmatrix}0 & -3 & 2 & 3 & 4\\ 3 & 0 & -5 & 2 & 3\\ -2 & 5 & 0 & -7 & 2\\ -3 & -2 & 7 & 0 & -9\\ -4 & -3 & -2 & 9 & 0\end{pmatrix}\begin{pmatrix}p_1\\ p_2\\ p_3\\ p_4\\ p_5\end{pmatrix}\begin{pmatrix}0\\ 0\\ 0\\ 0\\ 0\end{pmatrix}$$

针对 Andy 的期望收益矩阵，由于其行列式为 0，且存在无穷多个方程集，因而需要添加额外的信息。鉴于概率之和为 1，因而 Andy 的信息中总包含 0 期望支付额度，且无须关心 Beth 的行为。当然，相同的方案同样适用于 Beth。相应地，对于非对称矩阵，则无须计算期望支付额度，且需要对 Beth 进行单独求解，同时包含了多个未知支付额度值。

【提示】Undercut 游戏的求解过程留予读者以作练习，对此，读者可参考第 3 章所讨论的联立线性方程组的求解方法。

Undercut 游戏的求解方案适用于任意零和游戏的分析过程，对于两名玩家而言，全部信息均为已知。对于非零和游戏或包含不完全信息的游戏而言（例如游戏 Poker），情况则变得越加复杂，读者可参考相关理论书籍以获取更多内容。

随着游戏的不同，玩家信息的获取方式也有所变化。同步博弈型游戏并不了解另一名对手的情况，例如前述囚徒问题。另一种游戏则是序贯博弈（sequential game），其中，各玩家仅知晓其他对手的部分信息，各玩家依次轮流选择，直至一方获胜。序贯博弈涉及较为复杂的工作，即马

尔科夫链。从计算角度上看，该方案相对特殊且涉及较少的数学内容。

25.2.3　Tic-Tac-Toe 游戏中的博弈论

类似于石头-剪子-布游戏，Tic-Tac-Toe 游戏同样广为人知，该游戏较好地体现了 AI 理论，如图 25.5 所示。其中，玩家 Beth 在第二列处置放了 3 个“O”，并以此获胜。在序贯博弈中，Andy 和 Beth 两名玩家交替进行，并在网格正方形中放置符号，直至网格填满或一方在一行、一列或对角线方向填满 3 个符号。

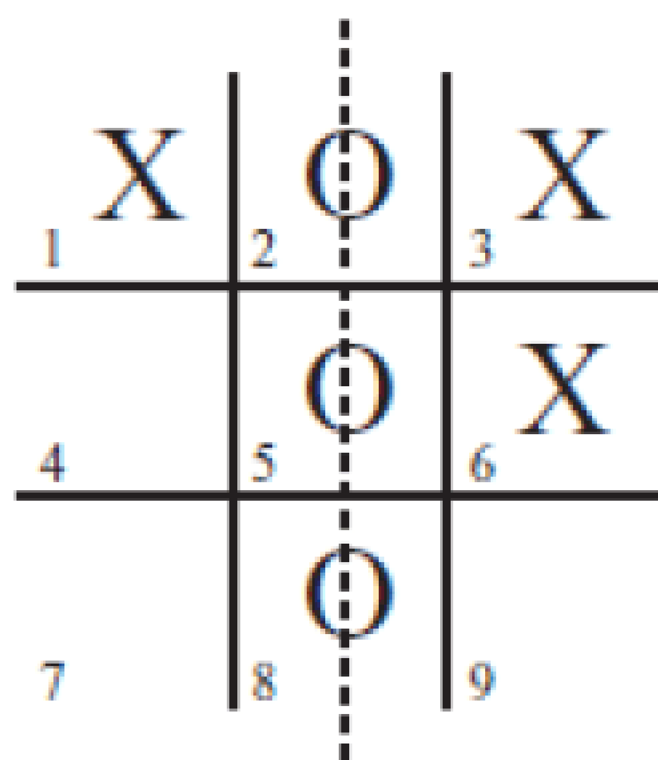


图 25.5　Tic-Tac-Toe

在图 25.5 中，正方形分别予以标记以便于参考引用。为了进一步考察博弈论与 Tic-Tac-Toe 游戏间的应用方式，首先需要查看最小-最大值这一概念，并对应于最小-最大算法。为了正确表达游戏体验，可生成相应的搜索树。搜索树的工作方式与八叉树、四叉树以及迷宫网络类似，树中的各游戏位置通过节点表达，各个节点则通过邻接移动予以连接。图 25.6 显示了顶端搜索树的状态，自上至下的各层（级）表示为某一玩家的移动行为。

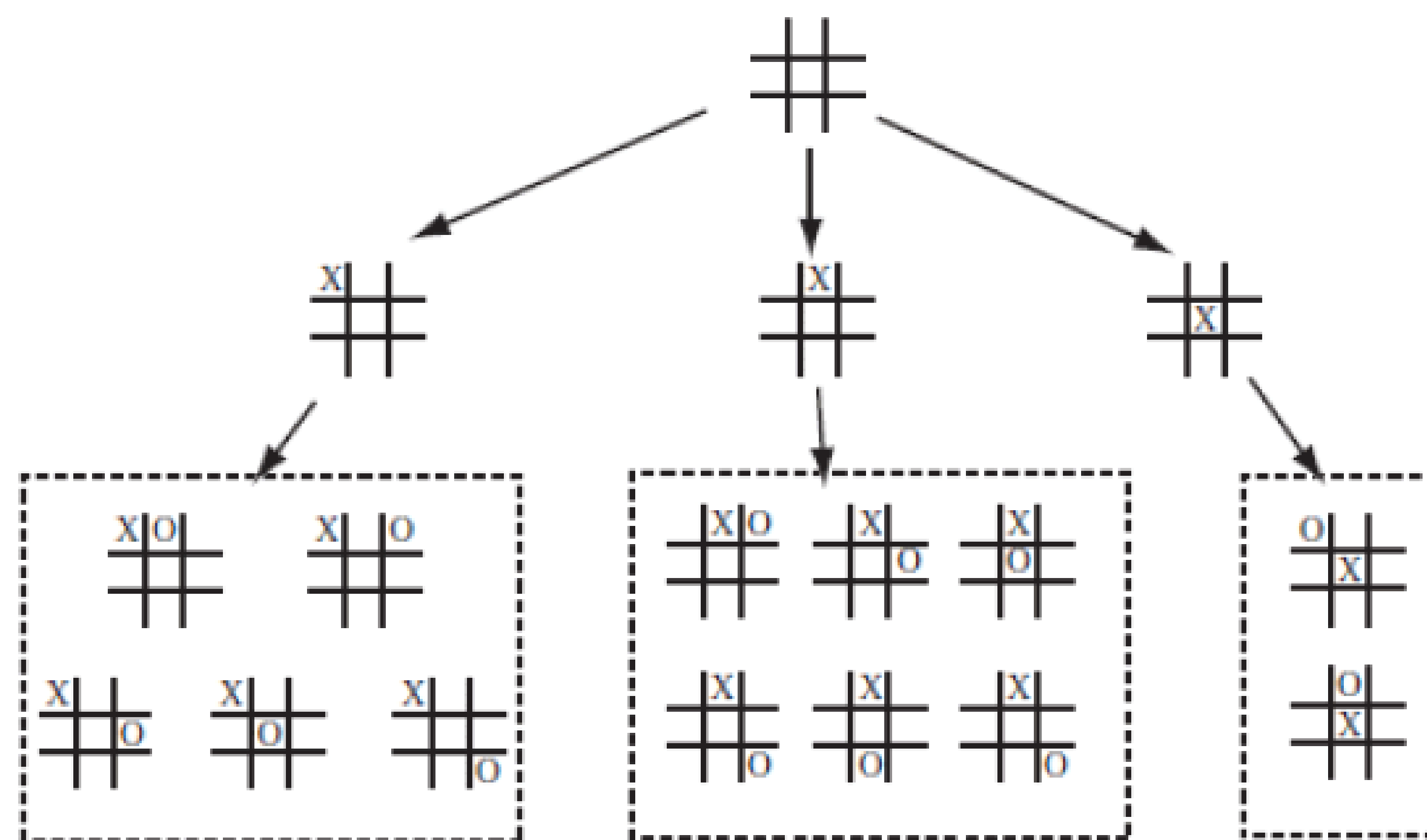


图 25.6　Tic-Tac-Toe 游戏中搜索树的开始阶段

【提示】术语“搜索树”并非严格意义上的树形结构，在某一层中，两个或多个位置也可能为下一层中的同一位置。

针对图 25.6 所示游戏，应注意优化措施。考虑到游戏的对称性，多个位置具有等效性。Andy（使用“×”标记）开始时仅包含 3 个不同的移动位置，即角、边以及中心位置。

为了有效地实现最小-最大算法，可于开始阶段遍历树形结构，并获取全部结束位置。这里，结束位置是指一行中包含 3 个符号的对应位置。对于玩家 1，获胜位置得分为 1；对于玩家 2，获胜位置则得分为–1；而其他绘制后的位置则记为分值 0。随后，可从此类结束位置沿树形结构遍历相关链接，通过此类链接可到达其他结束位置。若上述情形出现于树形结构的 Beth 层中，则可用节点最小值标记各节点，进而表明 Beth 是否可根据该位置获胜。若上述情形出现于树形结构的 Andy 层中，则可用节点最大值标记各节点。图 25.7 显示了包含此类计算值的搜索树的部分结构。

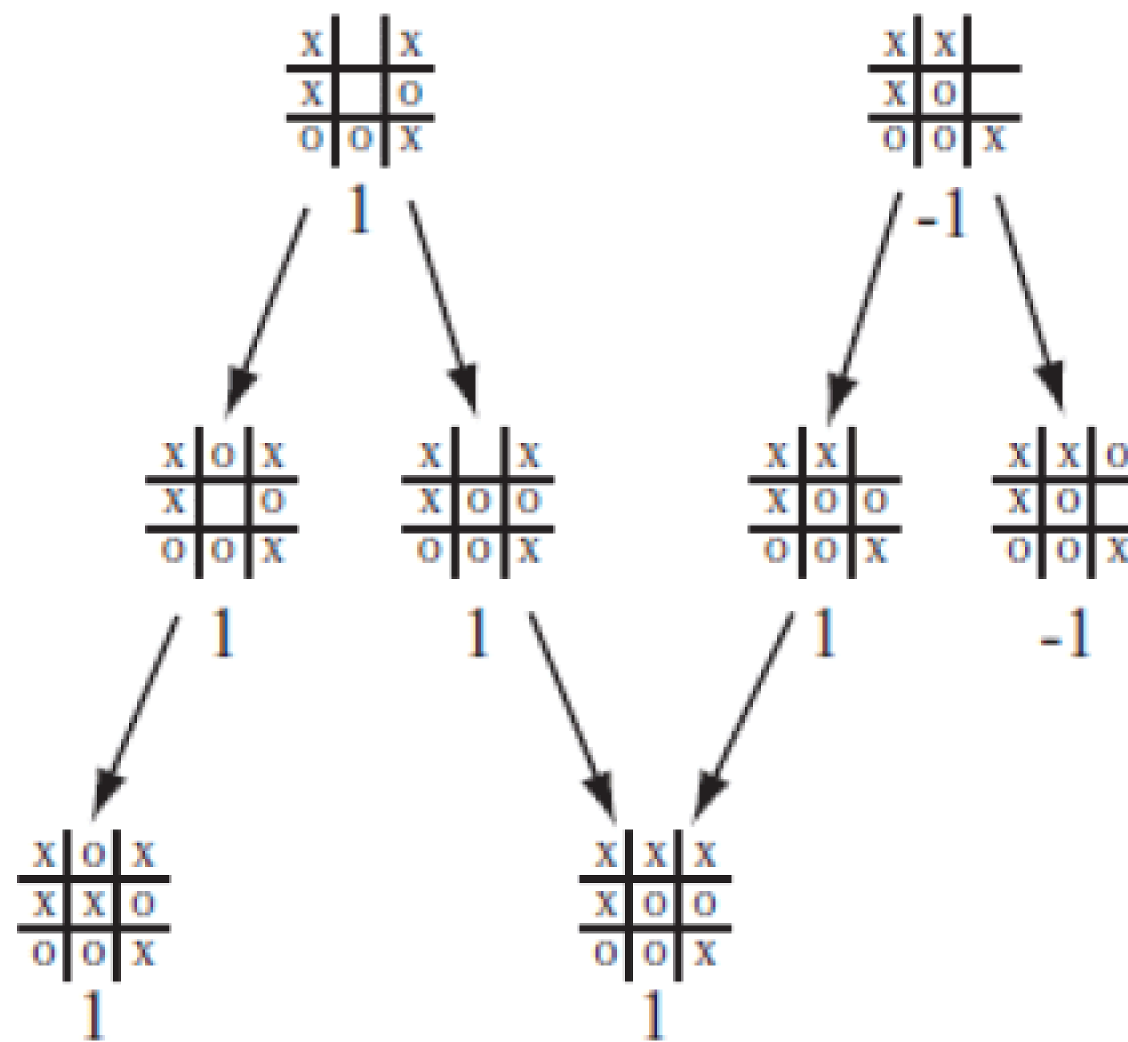


图 25.7　标记搜索树中的节点

25.2.4　Tic-Tac-Toe 游戏的搜索方案

根据前述讨论，大致可获得 Tic-Tac-Toe 游戏的完整功能集，并可得到 5 个函数，进而计算该游戏的求解策略。其中，makeTTTList()和 makeTTTtree()函数负责构造搜索树，其他 3 个函数则实现了分析型遍历功能。鉴于当前示例仅体现了 Tic-Tac-Toe 游戏的理论实现方案，因而暂不考虑其计算速度和内存使用状态；同时，通过保留游戏的对称位置，可进一步节省工作量。makeTTTList()函数如下所示：

```
functionmakeTTTList()
  //creates a list of all possible game boards
  setblist to an empty array
  repeat for i=0 to 5
    set blank to an array of 9 0's
    if i=0 then add blank to boardlist
    otherwise
      setbl to boardlist(blank, 1, i, 1)
      //list of all boards with i 1's
      repeat for j=i-1 to i
        if j=0 then next repeat
        repeat for each b in bl
          set bl2 to boardlist(b, 2, j, 1) //i 1's and j 2's
```

```
          add all of bl2 to bl
        end repeat
      end repeat
    end if
    end repeat
  returnblist
end function
```

makeTTTtree()函数如下所示：

```
functionmakeTTTtree(blist)
  //creates a tree of all possible moves in blist:
  //for each node, creates a list of all possible parents and children

  set tree to an empty array
  set e to array(empty array, empty array)
  add as many copies of e to tree as the elements of blist
  repeat for i=1 to the number of elements of blist
    set b to blist[i]
    //find all parents of b
    if b has no '1's then next repeat
    if b has an odd number of '0's then set s to 1
    otherwise set s to 2
    repeat for j=1 to 9
      if b[j]=s then
        setp to a copy of b, replacing the s with 0
        set k to the position of p in blist
        append i to tree[k][2] //children of i
        append k to tree[i][1] //parents of k
      end if
    end repeat
  end repeat
  return tree
end function
```

boardList()、makeMinimaxStrategy()和 minimaxIteration()函数负责计算获胜策略。boardList()函数如下所示：

```
functionboardList(board, symbol, n, start)
  //fills a board with n copies of the symbol
  //in all possible ways (recursive)
  if n=0 then return array(board)
  setbl to an empty array
  set c to the number of 0's between board[start] and board[9]
  repeat for i=1 to c-n+1
    set b to a copy of board
    set k to the position of the next 0
    set b[k] to symbol
    setbls to boardlist(b, symbol, n-1, i+1)
    append all elements of bls to bl
  end repeat
```

```
  returnbl
end function
```

makeMinimaxStrategy()函数如下所示：

```
onmakeMinimaxStrategy (tree, bl)
  //'prunes' tree by removing all unnecessary children,
  //and returns an initial strategy and minimax tree
  set strategy and minimaxtree to arrays of the same length as bl
  set each element of strategy and minimaxtree to "unknown"
  repeat for i=1 to the number of elements of bl
    set b to bl[i]
    if b is a win for 1 then
      set strategy[i] to "WinX"
      setminimaxtree[i] to 1
      deletechildren(tree,i)
    if b is a win for 2 then
      set strategy[i] to "WinO"
      setminimaxtree[i] to -1
      deletechildren(tree,i)
    if b is full then
      set strategy[i] to "draw"
      setminimaxtree[i] to 0
    end if
  end repeat
  return [strategy, minimaxtree, tree]
end function
```

minimaxIteration()函数如下所示：

```
onminimaxIteration(strategy, minimaxtree, tree, bl)
  repeat with i=the number of elements of bl down to 1
    //ignore nodes with no parent
    if tree[i][1] is empty and i>1 then next repeat
    //ignore nodes that have already been calculated
    ifminimaxtree[i] is not "unknown" then next repeat
    set c to the number of 0's in bl[i]
    set ply to mod (c,2)
    if ply=0 then set ply to -1

    if there is any j in tree[i][2] such that minimaxtree[j]=ply then
      setminimax to ply
      set mv to j
    otherwise
      //find best non-winning move
      setminimax to "unknown"
      repeat for j in tree[i][2]
        set m to minimaxtree[j]
        if minimax="unknown" then set minimax to m
        if ply=1 then set minimax to max(m, minimax)
        otherwise set minimax to min(m, minimax)
```

```
      ifminimax=m then set mv to j
    end repeat
  end if
  set strategy[i] to mv
  setminimaxtree[i] to minimax
 end repeat
 return strategy
end function
```

25.2.5 限制条件

尽管 Tic-Tac-Toe 系统可提供相对稳定的获胜策略，但依然无法适应各种场合，其原因源自相对低下的计算效率。即使对于简单的 Tic-Tac-Toe 游戏，也存在超过 6 千个可能位置；若消除反射和旋转行为，也将存在不低于 1 千个可能位置。对于棋类游戏，这一数字还将会成倍增加。针对各位置，至少存在 30 种移动方式，对应搜索树及其在游戏中的应用均需要占用大量的计算时间。

对此，存在多种方式可对搜索树进行优化，其中较为重要的是 alpha-beta 搜索。对此，当搜索特定深度的移动方式时，可对两个数值予以跟踪，即 alpha（最佳分值）以及 beta（阻止行进的最差分值）。

作为 Tic-Tac-Toe 游戏的 alpha-beta 搜索示例，假设 Andy 分别填充了方格 3 和 8，Beth 填充了方格 6，且轮到 Beth 行棋。当前，Beth 包含 6 个可选方格并假设移至方格 5 这一中心位置处。从理论上讲，Andy 可能并未发觉威胁，且未填充方格 4。在实际操作过程中，该情况并不会出现。相反，Andy 会走出正确的一步并对 Beth 进行封锁。这里，由于假设各玩家均选择最优步骤，基于最优方案的行棋缺少应有的自然性，还需要进一步计算相应的最坏情况。alpha-beta 搜索意味着全部重点均落于最优步骤上。

若途径某一节点其值小于当前 alpha 值，鉴于之前已获取一个较优节点，因而可忽略该节点。类似地，若某一节点的 beta 值大于当前节点，由于不会选取此类节点，因而同样可忽略该节点。

根据当前讨论，当模拟人机游戏时，对手的完美表现往往缺乏应有的真实感。实际上，此类假设条件可视为基于博弈论理性模型的主要问题之一，对应模型涉及某些经济学问题。然而，构造智能玩家的主要目标之一便是避免蛮力方案，其中最重要的是分析棋盘并选取相应的移动步骤。特别地，可执行某种预分析方案并检测期望路径，进而降低或简化搜索空间。

25.3 战术型 AI

玩家在游戏过程中，通常较少思考博弈论这一类知识。相反，玩家更关注于获取领地、开局设置、威胁性进攻以及机动调遣，即战术策略（而非战略策略，稍后将对此进行讲解）。战略策略关注于长期目标规划；战术策略则考察玩家移动时目标的实现或放弃，并涉及实时游戏体验。

25.3.1　棋类游戏的工作方式

1997 年，计算机“深蓝”战胜了国际象棋世界冠军 Gary Kasparov，这一重要的历史时刻体现了半个世纪以来 AI 程序设计方面的不懈努力。

国际象棋是一项复杂且难以处理的智力游戏，类似于 Tic-Tac-Toe，两名选手均可得到与游戏相关的全部信息，且不存在任何随机性。其中，有效步骤的数量十分庞大且令人难以想象。据此，程序开发人员很快认识到，国级象棋的搜索树几乎难以实现，并将重点转至理解选手的行棋方式，进而将此制定为行棋规则。

占领实地可视为国际象棋中的重要内容，大师级选手往往可准确、快速地对此进行规划，并将重点工作放在位置的争夺中。当棋子随机散落于棋盘上时，业余选手与大师级选手的处理手法基本相同。另外，当后者出现错误时，往往是大局观出现了问题，错误的行棋通常预示着危险的到来。

获取实地往往需要根据选手间的当前战术优势进行位置分析，而非计算获胜位置，并涉及估算函数。从根本上讲，估算函数等价于 A*算法中的测算函数，并可向某一特定位置赋予一个数字。该方案与搜索树进行整合时，可限制相应的搜索深度。最终结果可描述为：针对特定节点，可用估算函数的结果替换相关值（分别对应获胜/失败/平局）。随后，可继续采用 alpha-beta 算法。

估算函数和搜索结合使用可生成更为高级的方案，例如，可执行首级（层）移动的快速估算；针对第二级（层），则可丢弃大部分内容进而生成较小的计算集合、当移动行为再次被剔除后，仅需通过少量步骤即可得到主要路径，这也与真实的选手行棋方式保持一致。高水平选手并不会分析全部的可能步骤，或者是一些错误的步骤，即使是业余选手也不会执行后退或令车以对角线方式行进这一类错误的走法。

估算方案不可避免地限制了搜索深度，由于某些获胜路径在初始阶段貌似错误，因而最终难以被发现。例如，基于估算函数的程序难于发现弃子获胜时的路径规划，与长期目标相比，当下得失则更易于感知。当然，若计算机算法失效，则选手同样会面临此类问题。

25.3.2　程序训练

估算函数的计算过程涉及大量的预测工作、反复试验以及某些数学分析实际上，相关工作可归结为一点，即需要计算一组可测量的参数集，可简单地描述当前位置，并可对获胜几率产生影响。多个参数均可应用于棋类程序中，例如各选手全部棋子数量、处于威胁状态下的棋子数量。除此之外，其他内容还包括：中央位置的控制、王的置身状态、棋局中相关棋子的状态以及兵的作战力量。

可测参数的另一个优点是可高效地处理游戏中的随机元素，例如西洋双陆战棋。其中，即使无法确定连续轮次采用的行棋步骤，玩家依然可计算当前位置的力度、棋子的布局方式以及将被吃掉的棋子的数量等。

为了将估算参数转化为计算表达式，待参数认定完毕后，可于随后构造此类数字的加权求和

结果，即 $a_1p_1+a_2p_2+\cdots+a_np_n$。其中，$a_1a_2$，…均通过预定义方式加以确定。

作为程序设计人员，上述参数化过程不可避免。鉴于理论上无法获取正确的权值，因而须通过反复试验这一方式对其进行计算。其中，自然选择算法可视为一类最为高效的方法，第 26 章将对此予以讨论。除此之外，另一种计算方法是使用训练系统。通过分析特定移动的成功/失败结果，程序可适当地调整所用权值。

对于较为简单的训练系统，可实地考察游戏程序。其中，两个训练系统分别有两名选手。当采用某一权值系统时，训练系统可列举出当前位置值，随后，可通过固定数量的移动步骤进行预测，进而选择包含最大化期望值的对应步骤。此时，对方程序同样会选取移动步骤，之后，当前选手的程序将再次启动。若基于当前位置的估算函数的计算结果大于或等于程序期望结果，则移动操作可行，则可稍稍增加当前决策中的权值。若当前值小于期望值，则移动行为无效，并增加对应参数的权值。

训练和学习算法与操作条件有关，作为一种学习形式，操作条件采用了奖励和惩罚机制，且多与心理学家 B. F. Skinner（1904—1990）联系在一起。当使用操作条件时，某一行为根据以往的成功经历得到加强。另外，基于神经网络的程序将会使用到与操作条件相关的符号。

25.3.3　基于 Tic-Tac-Toe 游戏的战术 AI

关于训练程序与 Tic-Tac-Toe 游戏之间的应用方式，建议先期针对潜在可行的估算参数制定对应的候选者，下列内容显示了几种可能性：

（1）空行、列或对角线的数量。

（2）仅包含当前玩家符号的行数。

（3）仅包含另一名玩家符号的行数。

（4）潜在“X”符号的数量（包含当前玩家符号的交叉行）。

（5）源自另一名玩家的潜在“X”号数量。

（6）潜在的威胁数量（包含当前玩家符号和空正方形的行）。

（7）源自另一名玩家的威胁数量。

上述参数列表包含了某些冲突项，针对前述 Tic-Tac-Toe 游戏，若 Andy 占据了上方中心位置，而 Beth 填充了下方中心处——Beth 封锁了 Andy 的某一潜在行，因而通过上述方案三得分。另外一方面，针对底部边角处，Beth 同样可根据方案二得分。那么，两种方案孰优孰劣？此处并不存在一个明显的答案，对此，可采用训练系统计算包含较高权值的方案。其中，多种方案可赋予 0 权值，即对应方案对最终结果不产生任何影响。除此之外，还可计算不同的权值集，并生成异曲同工的战术体验效果。

25.4　自顶向下型 AI

若不采用战术处理方案，还可通过战略方法处理 AI 问题。换言之，可使用某一系统对自身

设置某个目标并尝试对其予以实现。这里，制定目标并实现目标也称作自顶向下型方案。其中，高层决策也用于制定底层处理。某些时候，缩略词 GOFAI（good old-fashioned AI）常用于自顶向下型方法中，即具有较好效果的传统型 AI。这一称谓历时已久，且多用于心理模型。科研人员通过符号结构描述当前世界，并构造基于事实和决策制定的推理模块。近期，该方案在认知科研工作者中逐渐势微，并逐渐被自底向上型方案所取代。尽管如此，在有限的某些领域内，以及诸如游戏这一类专家问题中，自顶向下型方案依然十分流行。

25.4.1　目标和子目标

基于目标的 AI 处理方案类似于程序设计过程，问题通常始于某一项较为困难的任务（例如制作一款第一人称射击游戏），随后，可将该任务划分为多个子任务（包括主要角色、场景环境以及敌方角色）。该划分过程可持续进行，直至对应任务单元可直接编程实现。

上述处理过程的难点在于如何确定子任务，其中，获取特定目标的相关步骤通常并不明显。在棋类游戏中，将“将杀对方的王”这一任务划分为多个子任务通常十分复杂，程序需要使用到知识库。这里，知识库表示为一类场景描述，并可用于制定相关策略进而做出相应判断。例如，棋类程序应包含某些启发式策略，其中包括“避免王被将杀”、“尝试先期防御”等内容。知识库同样包含某些推理机制，尤其是推测机制。例如，当采取防守措施且皇后将杀时，此类机制应对皇后予以保护。

待系统配备了知识库后，程序将规划其战略方案、制定特定的情景、构造战略表达方案并据此进行处理。首先，程序将对当前目标进行评估（例如利用 e8 上的皇后将杀王），查看基于该目标的潜在危险（例如 g7 中的象移动后将对其进行遏制），进而生成子目标（消除象造成的威胁）。同时，程序还将针对当前子目标寻求解决方案（利用马捕捉象），此类行为持续进行。在处理过程中，程序关注于单一步骤并推迟大多数中间子目标。

对于棋类游戏而言，目标程序与前述战术 AI 相比更接近于人类的行棋方式。因此，AI 研究人员在早期对自顶向下型方案抱有更高的期望。实际上，该方案具有显著的效果，尤其是专家系统领域。然而，考虑到知识库的需求规模，目标方案难以扩展至动态领域，例如自然语言。

25.4.2　改变目标的时机

由于各种规划方案均存在潜在缺陷，因而战略与战术安排彼此关系紧密。没有人可预测一切，计算时间的限制条件意味着各策略间需包含一定的“空白”位置。实际上，战略思考的主要目标之一即是无须担心最后一步的计算过程。这里，选手只需朝最终目标方向移动位置。在棋类游戏中，选手不必关注皇后是否受到了马或象的保护，仅需了解总体规划是否成功，进而以某种方式保护皇后。然而，此类细分操作可导致某些意外步骤阻碍当前规划。

一类有效的战略和战术整合方式是，在战术程序的估算函数中将战略规划视为权值集合，该方案在棋类游戏中表现稍差，但却适用于某些简单的游戏，例如西洋双陆战棋。此处存在两种战略规划，规划一可描述为：构建强势局面保护己方棋子，并捕获敌方棋子。当局势恶化时，则可

调整至不同的策略，即规划二。在规划二中，通过弃子可获得相应机会以遏制对方，并可动态逆转局势。当然，该策略具有一定的风险，但场面也变得更加刺激。

相应地，可通过基于变量的估算函数对上述两种规划进行建模，在各阶段中，函数将在当前规划下生成最佳值。若该值低于某一既定阈值，则程序尝试其他权值集合；相反，对于行棋步骤产生的较大值，程序则切换至其他战略规划。

当与策略型程序对战时，可根据其行为方式制定相应的策略。对此，策略游戏中另一个较为重要的方面则是模式分析。当采用模式分析时，可针对玩家最近几次行棋方式确定相应的应对策略。网络中提供了大量的基于模式分析的游戏示例，例如石头-剪子-布游戏，由于人类并不擅长于随机生成数字，因而此类游戏的取胜难度较大。其中，程序搜索前几次操作模式，并对随后步骤进行推测。

最为简单的模式分析系统将记录 3 次行棋步骤，例如 RPRSSRP。对此，程序将获取 RPR，PRS，RSS，SSR，SRP。对于 RP，则玩家很可能出具 R。随着游戏轮次的不断增加，程序将推断出玩家仅出具 RP，且 P 的几率将 2 倍于 S，并于随后猜测这一模式将会持续进行。总体而言，玩家的游戏时间越长，其行棋力度将越发低下，一些更为高级的系统还将对上一轮的比赛结果予以记录，在练习 25.1 中，读者将尝试编写此类程序。

对于更为复杂的游戏，模式分析依赖于程序的策略模块，并涉及以下问题：当前环境下如何执行下一步操作？对此，程序可估算对手各种不同行为的可能性。例如，相关行为可分为进攻、防守等内容，这将有效地改善搜索树的剔除操作。

脚本系统可视为压力测试下目标变化的另一个示例，该系统采用预定义策略直至出现非期望事件，例如隐蔽类游戏中的机器人。其中，机器人以标准模式或包含特定参数的随机模式运动，直至其觉察到声音或发现尸体。随后，机器人的脚本将发生变化，进而切换至不同的预警模式或搜索对应策略。相应地，新脚本将包含“警告其他人员”或“搜索入侵者”等目标。

25.4.3　Tic-Tac-Toe 游戏的自顶向下 AI 方案

为了重申早期定义，自顶向下型 AI 在底层获取源自高层的决策信息，同时，原目标演变为子目标。当引用树形分析时，Tic-Tac-Toe 中的主要子目标为“X”形符号，该符号可视为赢取游戏的唯一方式，“X”形符号下方则是获取控制权的空行。此类目标与构造估算函数的参数保持一致。由于战术有效性同样适用于战略方案，因而当前搜索路径可行。然而，在某种程度上讲，当前处理方案依然是一类简化结果。在更为复杂的领域中，则存在相应的描述级别并超出了估算函数的计算范围。例如，在棋类游戏中考察下列情形：皇后是否可到达方格 b5 处？该问题的有效性取决于当前目标，对于大多数决策而言，由于这构成了估算参数的无效附加条件，因而可视为一类无关问题。

在 Tic-Tac-Toe 游戏中，主要目标是生成“X”形符号，但子目标很可能是在左侧直线和上方直线之间构造“X”符号。对于此类子目标，问题也随之而来：上方数据行是否为空？若采用估算函数，则该问题不存在有效性；若转换为策略型，则问题可迎刃而解。

为了深入讨论这一问题，可参考图 25.8，图中采用相关数字标记移动行为，进而整体展示了 Tic-Tac-Toe 游戏。

6 O	5 X	1 X
9 X	2 O	8 O
3 X	4 O	7 X

图 25.8　策略型 Tic-Tac-Toe 游戏

在游戏开始时，Andy 面临一种空白棋盘，其目标生成组件开始搜索可能存在的“X”号。最终，Andy 决定在角方格 3 处标记“X”号，此类决策通常根据概率进行选取。根据 Andy 的知识库，为了获取较好的“X”符号路线，此处选择了角方格 7，进而强迫 Beth 移至该行的中心位置，从而留下其他空白地带供其选择（例如在两个边侧行中标记“X”号）——上述内容即为 Andy 的当前策略。

然而，考虑到战术与战略（策略）思考之间的差异，多数场合下，可通过最优方案制定相关处理方法。若对手并未猜中行棋意图，则可思考下一步策略。在博弈论方案中，可根据最坏情况予以思考，并假设对手做出最坏的应对。当然，当确定最优策略时，依然需要考虑对手的反应。需要说明的是，此处不存在一类注定失败的策略。

同样，Beth 也了解行棋规则，因而会极力避免令自己陷入困境。鉴于最简单的遏制方式是对角封锁，因而 Beth 的下一步置于中心位置，这也有利于开创自身的局面，进而将 3 个标记连接在一起。

当前，Andy 的最初方案部分受到破坏，因而需要在原步骤基础上寻找其他路线。

由于 Andy 依然部分控制着上方和右侧方格行，因而仍存在回旋余地，并包含两种可行的下法，例如方格 7——虽然这是 Andy 的原目标，但当前意义已产生变化。

需要注意的是，Andy 的移动步骤已经破坏了 Beth 的策略，也就是说，Beth 所期望的下方一行已被占据，相应的替换方案则位于左侧，即位于中心和对角方向上的两行。进一步讲，若 Andy 未阻挡 Beth，则 Beth 难以发起有效的进攻，其唯一选择即是迫使 Andy 求和，这将演变为一个新的策略规划。通过快速预测可知，若 Beth 在边角处行棋，则 Andy 将同时进攻两个棋子，因而 Beth 需要在某一边侧处落子。自此，双方行棋进入预定模式，即两名选手彼此遏制对方。

尽管上述分析稍显过分，但却可在一定程度上丰富游戏的体验，并可有意识地在战略型和战术型方案之间进行调整。

25.5　自底向上型 AI

与自顶向下的目标驱动型程序员不同，连接机制赞同者多采用自底向上型方案，并通过大量的简单、迟钝元素的交互行为尝试构建智能行为。此类型系统常出现于人类大脑的工作模式中，即通过神经元实现交互操作。除此之外，蚁丘中也存在类似的行为，其中，蚂蚁的交互行为使得

蚁丘呈现出整体目标驱动所形成的外观。据此，功能、目标和决策均作为机械事件的高层表达加以展现（副现象）。当前，自底向上型处理方案较少出现于游戏中，鉴于其潜在优势，该方案依然值得进一步讨论。

25.5.1　神经网络

神经网络（或称作神经网）可视为纯粹的连接机制，且类似于大规模并行处理计算机，此类计算机由多个小型计算机构成（模块化于神经单元上），并称作神经元。这种智能型或模拟型神经元于网络中彼此连接，并包含了相应的输入或输出数据，通过学习处理过程训练后可生成特定的结果数据。

图 25.9 显示了神经网络的工作方式，其中神经元从根本上可表示为一类简单的计算设备，各神经元含有一个主单元体（或称作神经元胞体）。大约上千个单纤维（称作树突）构成了神经元胞体。对于输出数据，神经元包含一个轴突，并连接于神经元胞体一侧，轴突可分为称作“末梢”的各种单纤维。类似于树突，此处存在约上千个末梢，各末梢止于称作“终端钮”（terminal button）的结节处。其中，终端钮通过神经元突触连接至其他神经元的树突。

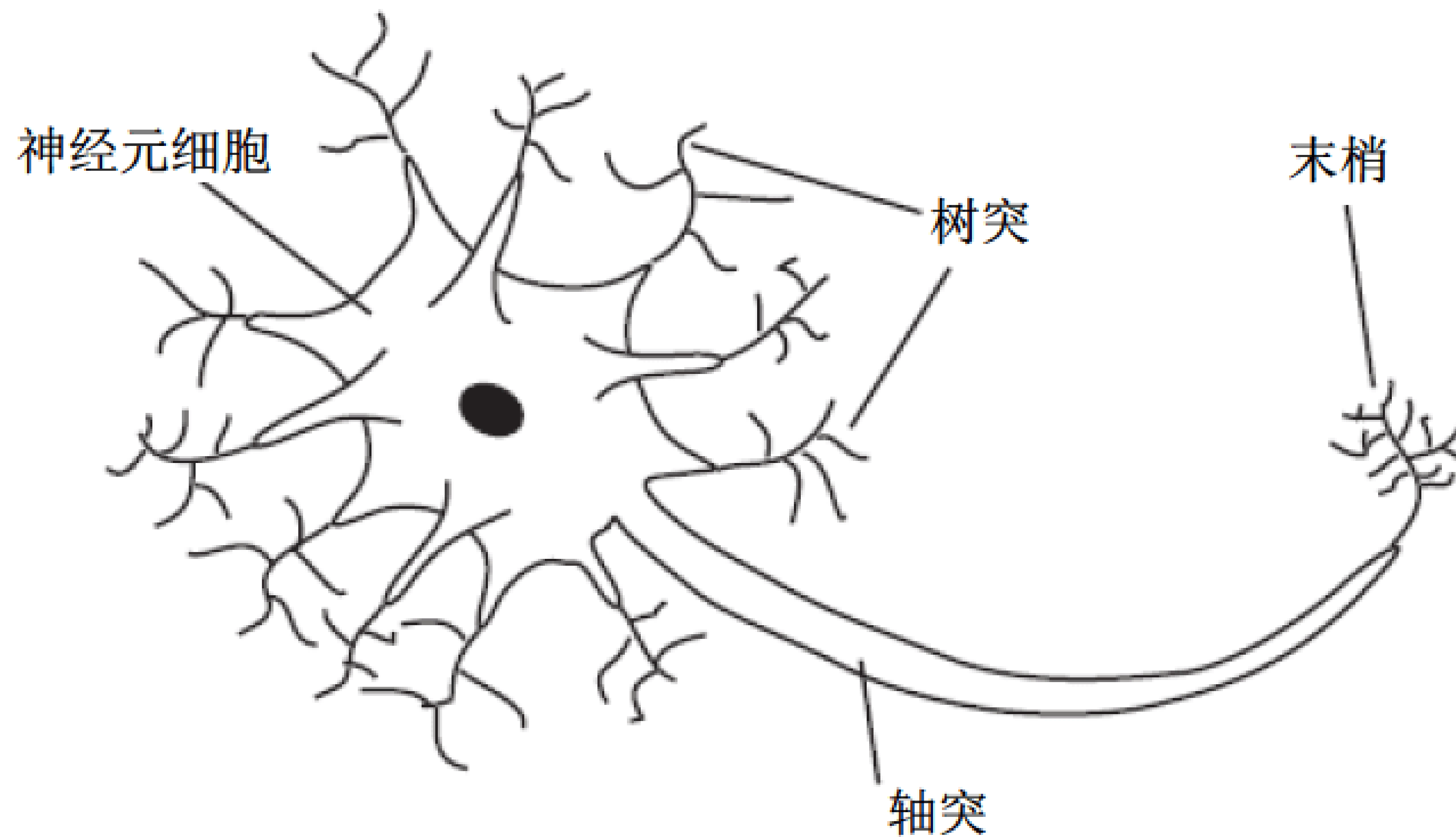


图 25.9　脑神经示意图

神经元通过化学-电子信号工作，并可于任意时刻沿轴突触发某种强度的信号。另外，固定的神经元通常会产生相同的信号强度，但信号频率一般会产生变化。然而，神经元是否于某一特定时刻被触发则取决于进入树突的信号。

任意时刻，若进入树突的全部信号大于某一阈值，则神经元被触发。根据树突的具体行为，其和将通过加权方式进行计算。若某些树突处于兴奋状态，则接收信号将被加入至全部求和结果中。相应地，其他树突则处于抑制状态，而信号则从全部求和结果中减除。除此之外，还存在一些附加处理过程，并将某些急促的弱信号解释为强信号。

待读者理解了神经元的工作逻辑后，研究人员将其用于计算科学中。为了构造神经网络，须对人工神经元进行建模。人工神经元缺乏生物神经元所包含的复杂性，树突、神经末梢以及其他元素的行为均被如图 25.10 所示的简单函数所替代。

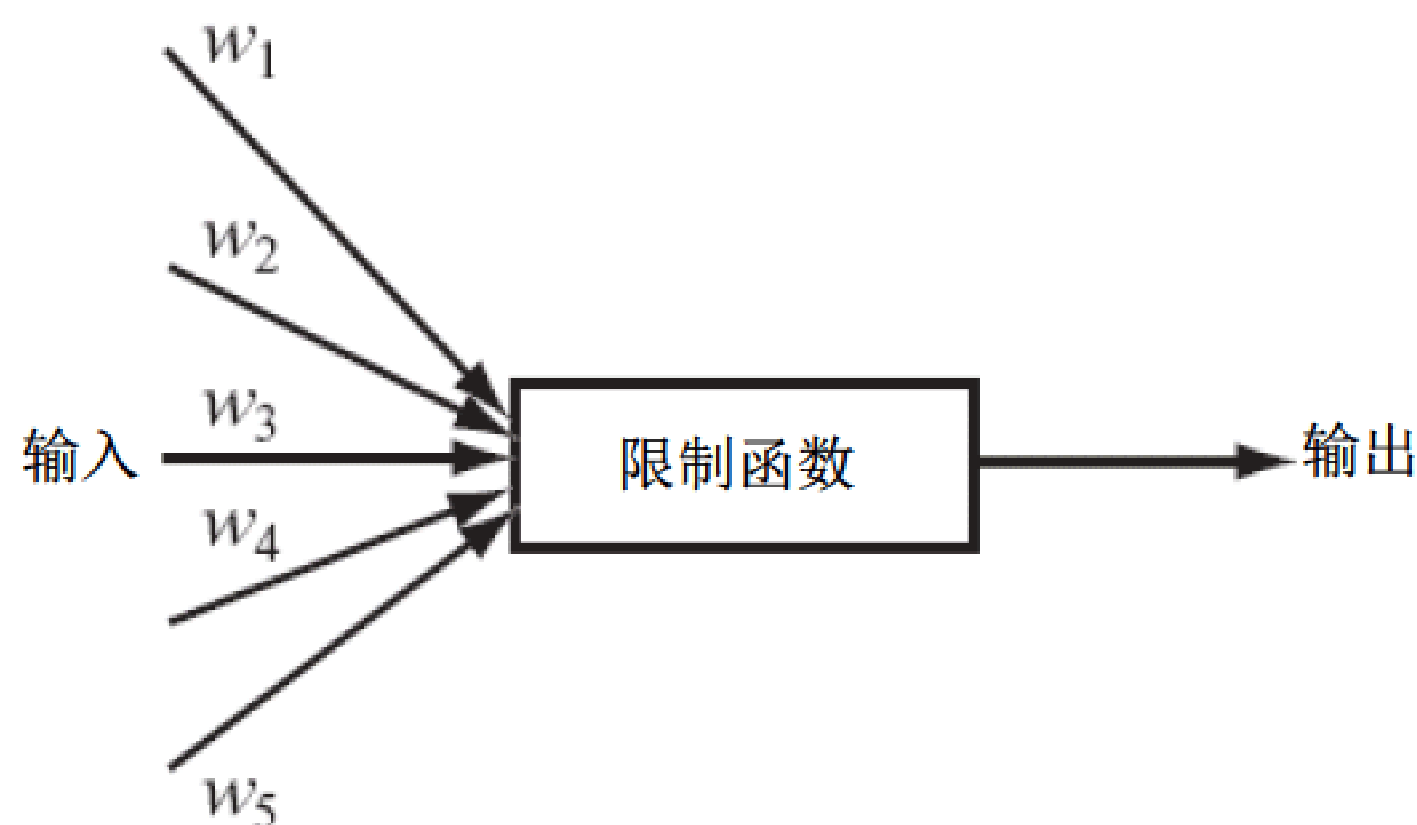


图 25.10　人工神经元

人工神经元包含特定数量的输入端，且均包含所关联的权值，并可赋予相应的阈值以确定是否可被触发。类似地，人工神经元也包含输出端，并可与任意数量的其他神经元进行连接。在神经元程序的各个时间步中，可计算输入端的加权求和结果，并查看是否超出了除法阈值，进而确定各神经元的输出强度。若神经元被触发，则输出结果为离散值 1、0，或者根据输入值的大小产生变化。

这里，可假设全部输入均传入至限制函数中，进而避免产生某些问题。其中，限制函数将输出值限定于 0～1 之间。针对基于阈值的离散输出值，该函数表示为步进函数，若输出结果大于特定值，则函数输出 1；否则，输出结果为 0。对于可变输出，则可使用 S 型（sigmoid）函数 $\frac{1}{(1+e^{-x})}$ 的某一个版本。上述内容实现于 neuralNetStep()函数中，如下所示：

```
function neuralNetStep(netArray)
  set nextArray to a copy of netArray
  repeat for each neuron in netArray
    set sum to 0
    repeat for each input of neuron
      set n to the source neuron of input
      add (weight of input)*(strength of n) to sum
    end repeat
    set strength of neuron in nextArray to limiterFunction(sum, threshold of neuron)
  end repeat
end function
```

待神经元网络构造完毕后，须创建某类型的连接网络。如图 25.11 所示，此处应构建神经元的输入层并连接至源。其中，源可表示为视觉显示或某一问题的数据值。除此之外，还可将输入层连接至输出层，而输出层可能为第二个视觉显示。同时，在输入层和输出层之间，还可设置一个或多个隐藏层，这将生成多层感知器（MLP）网络。

在图 25.11 中，网络可描述为计算面部的情感状态。其中，面部可分别表示为微笑状态和蹙眉状态。这里，输入节点的强度根据位图中像素的灰度颜色加以确定，对应结果则取决于独立神经元的输出强度予以定义，即 1（微笑状态）～0（蹙眉状态）。相应地，更为复杂的网络则包含更多数量的输出节点。

某一层中的各个神经元将连接至邻接层中的各神经元处，因而源自某一层的信息可传输至邻

接层中的各神经元中。需要说明的是，神经元信息在层间不可反向传递，这大大简化了网络的训练操作。当然，这与真实的大脑神经元相比其真实性也随之降低。

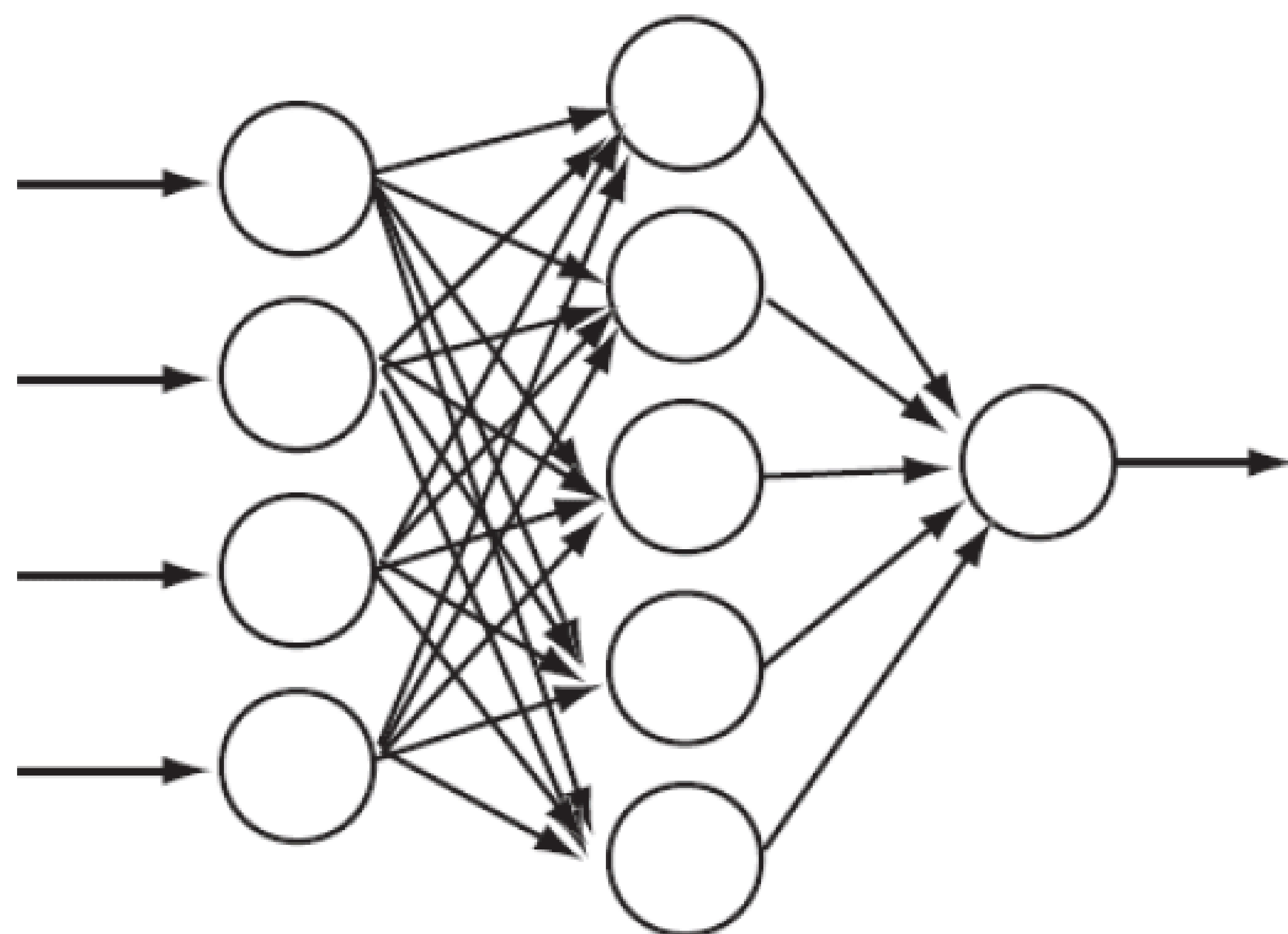


图 25.11　多层感知器网络

25.5.2　神经网络训练

针对特定问题，需要对网络训练以对该问题进行求解。此处，神经元网络对于即将求解的问题一无所知，且不存在显式的规程和规则。全部结果均整合自神经元的交互行为。最终，网络的训练操作相当于通过演化过程生成相应的计算机程序。

关于 MLP 或其他简单网络的训练过程，其核心内容称作反向传播。与前述 alpha-beta 搜索方案的训练方法不同，反向传播通过行为调节方式工作。具体而言，可向网络输入数据，对于接近期望值的输出结果，可对当前网络予以“奖赏”；而对错误的输出结果，则可对当前网络进行适当“惩罚”。其中，“奖赏”与“惩罚”机制由网络权值的变化构成，并可据此改善网络的传输结果。

当采用“微笑-蹙眉”检测器时，可对特定图像进行搜寻和训练。这里，假设将图像作为颜色值集合传递至输入层后，输出神经元所生成的数据值为 0.62，且最初的原始输出值为 1。此处误差为 0.38，因而须将该值沿对齐网络反向传入。

此处的理念可描述为：调整各层权值，以使其针对各次网络脉动生成较好的计算结果。开始阶段，可调整输出层的权值，以使其在接收输入数据时生成接近于期望值 1 的计算结果。该效果可反向应用于隐藏层，改变其权值进而生成与期望值接近的结果值。上述过程与前述训练操作类似，实际上，针对 alpha-beta 搜索方案的估算函数，神经元网络可视为一类有效的处理方法。

另外，也可通过演化机制训练神经元网络，并“结合”不同网络以生成子网络，且于随后根据其后代选择最优结果——这可视为第 26 章所讨论的通用算法的变化版本。

25.5.3　行为者和涌现性

与神经元网络相比，一类更为高级的连接机制则采用了称作“行为者”或“代理”的小型交

互子程序。该方案在面向对象程序员中较为流行。类似于对象，行为者表示为软件空间内的自包含实体，并可视为多个独立系统。行为者包含简单的行为，但考虑到将以多种不同方式与其他行为者进行交互，因而整体效果可生成较为复杂的高级行为。

广告牌模型可视为行为者的一个示例，其中，行为者通过在广告牌上发布信息实现交互行为，而其他行为者则可获取此类信息。同时，此类消息采用紧急程度进行标记，并包含了相关信息或须完成的目标，不同的行为者可能会使用到消息内容。另外一种类型的广告牌模型板报消息供全部行为者查看，行为者根据自身内容移除或调整现有消息。

涌现性（emergence）可视为全部连接机制的共有特征，并支持较高程度的组织性，且源自简单元素的交互行为。当考察大量的、独立的交互行为时，其价值方得以体现，例如交通、人群以及经济学。

群集行为即为涌现性的一个示例，并涉及鸟群、鱼群以及群集动物的运动行为。早期较具影响力的群集模型则是 Craig Reynolds 编写的 Boids 程序，该程序包含大量的简单生物，并遵循下列 3 项规则：

（1）移向其他生物的中心位置。

（2）与生物的平均速度匹配。

（3）生物间保持一定距离。

除此之外，还可添加其他规则，例如躲避捕食者或搜寻食物。即使仅采用上述 3 项规则，也可真实地体现群集行为，此类规则源自真实的物群运动。Boids 模型常用于计算机图形学、游戏以及电影中，例如《侏罗纪公园》（拍摄于 1993 年）。同时，更为复杂的行为者系统多出现于战争场面，例如《指环王》三部曲（拍摄于 2001—2003 年）。

最后一个方法则是 Douglas Hosftadter 所提出的高级感知方案，并采用各种自顶向下方式构建环境表达，该方案处于自底向上和自顶向下方案之间。另外，这一类系统采用了大量的称作 codelet 的代码对象搜索数据模式。codelet 则使用与 slipnet 关联的移相网络中的对应策略，后者包含了当前模块所理解的全部概念，例如同一性、差异性、互逆性以及组合性，进而构造特定场景画面。例如，“abc” 可在某一场合下视为字母的后继组合，而在另一种场合下则表示为长度为 3 的字母组合，此类混合感知能力具有一定的智能效果。

25.5.4　Tic-Tac-Toe 的自底向上 AI 方案

游戏 Tic-Tac-Toe 自身并未完美实现自底向上型处理方案，当然，神经元网络可使用基于 alpha-beta 搜索的估算函数。若采用此类方案，则处理结果难以得到有效改善，且需要针对行为者的广告牌位置进行分析。然而，这并不表示自底向上型方案缺乏应有的吸引力，与其他方法相比，该方案仅无法有效地生成智能程序而已。

一类有效方案涉及构建神经元网络，并使用任意广告牌位置同时返回下一位置。该方案采用一类稍显不同的训练机制，也就是说，未使用独立激励方案进行表达进而计算反馈结果。相反，该方案将体验全部游戏，从而根据输赢结果予以计算。

25.6 本 章 练 习

【练习 25.1】试编写模式匹配程序，以实现人机之间的掷币游戏。对此，可使用本章所讨论的任意方案，其中，自顶向下型 AI 可视为一类最为简单的方法。

【练习 25.2】试生成群集模拟环境。本章所描述的 3 个规则易于实现，并可生成令人满意的运行行为，在 3D 环境下尤其如此。除此之外，读者还可尝试添加捕食者躲避规则。

25.7 本 章 小 结

本章涉及较多内容，包括通用 AI 处理方法及其针对不同问题的应用方式。尽管本章并未包含 AI 实现的足够信息，但却指出了与此相关的考察方向。

第 26 章将继续深入介绍本章内容，并阐述通用算法以及大型空间的搜索方案。

至此，读者应掌握如下内容：

- 基于博弈论的简单游戏分析方法。
- 如何构建小型游戏的整体策略。
- 如何通过 alpha-beta 搜索机制简化某一规划策略。
- 估算函数的含义及其训练方式。
- 自顶向下和自底向上方案之间的差异，以及强 AI 和弱 AI 之间的差异。
- 如何将程序划分为目标和子目标。
- 神经元网络的构造方式以及问题的求解方式。

第 26 章　搜 索 技 术

本章包含如下内容：

- 概述。
- 问题求解方式。
- 用例学习。
- 遗传算法。

26.1　概　　述

本章通过计算机求解相关问题，进而扩展前述章节所讨论的内容。针对特定问题的解决方案，或包含某一特定结果的相关问题，本章主要关注计算机的应用方式，并针对各种可能性实现搜索。在第 22 章曾有所提及，尽管某些低效算法可对问题进行求解，但获取最优方案通常较为困难。在第 24 章曾讲到，可通过某些技巧获得较好的计算结果。除此之外，针对一些冗长的搜索行为（可能多达数天），读者仍需了解计算机在数据资源方面的组织方式。

本章讨论的大多数技术与前述章节中的内容并无太多区别，AI 问题与高级搜索之间并不存在严格的界线。然而，二者的应用领域却截然不同，因而有必要考察不同场合下类似方案的运用方式。

26.2　问题求解方式

本小节主要讨论通用处理方案，其中，相关问题可转换为计算机可处理方式，并主要集中于计算能力和搜索空间问题。

26.2.1　问题表达

基于计算机的问题求解过程主要涉及两个应用领域，游戏环境尤其如此。作为一名游戏玩家，当求解谜题且不知所措时，这一问题体现得尤为明显，例如回文构词生成器可在填字游戏中获取有效的答案。不甚明显的一面则是，游戏制作者使用计算机针对特定的答案制定谜题。例如，通过单词填写纵横网格，或者构造棋类问题。此两种情形基本类似，但填字游戏显然更为有趣，因而本章将着重对此进行讨论。

对于填字游戏，首要步骤是尝试对搜索空间进行分类。这里，搜索空间是指当前问题可能的答案集。当填写如图 26.1 所示的文字游戏时，搜索空间表示为各空方格（或单元）中某一字母的全部可能组合。在当前示例中，搜索空间由 26^{25} 个可能的纵横字谜构成，因而可将其表示为 25 元素数组。

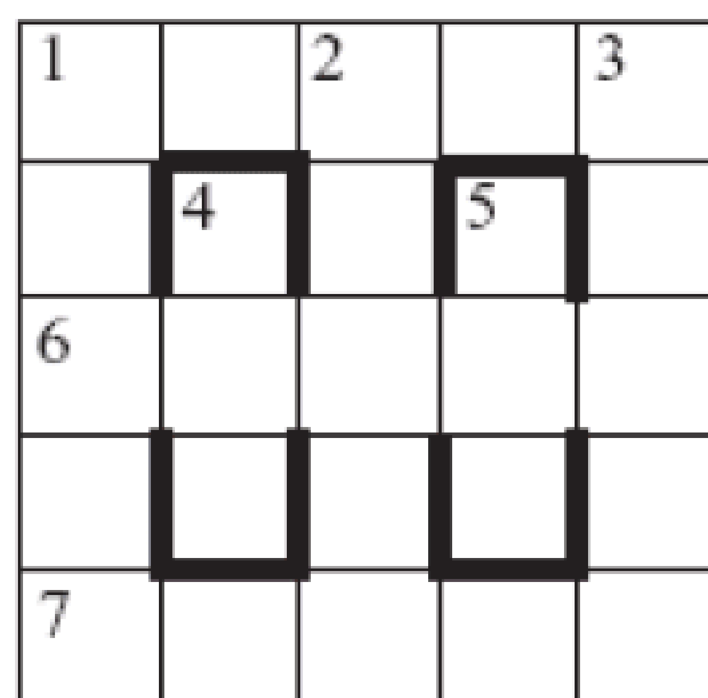


图 26.1　空填字游戏网格

然而，这并非是表达网格的最优方式：仅少部分数组体现了当前问题的有效解。一类更为有效的方法是表达相应的单词而非网格单元，这将产生 18 个元素的数组，且各项包含一个单词，因而更适用于搜索操作。然而，除了单词自身之外，仍需进一步考察单词之间的链接。对此，可在处理过程中求解方法附近处构造模板范本，在该范本中，各单词被命名并通过较小数组予以表示，如下所示：

```
2dn: [1ac/3; blank; 6ac/3; blank; 7ac/3]
```

当使用模板范本时，当算法填写某一单词的潜在求解方案时，可互相参照并检测该模板，并即刻进行填写。对于生成计算机所使用的当前问题的表达方案，此类初始考察方案至关重要，进而可执行相应的搜索策略。

26.2.2　搜索答案

在讨论某些搜索示例之前，下面首先讨论一类一般概念。计算机处理与手工求解之间的主要差别在于：前者设计较大的搜索空间，但对应问题相对简单。例如，图 26.1 中的填字游戏包含 10^{35} 种可能的字母组合。若期望快速获得计算结果，则搜索全部组合的计算阶数将十分庞大。对于任意字母组合，可查看各单词是否出现于字典内，进而可简单地对求解结果予以检测。

上述内容即为 NP 难题（这里，NP 表示非确定性多项式时间）。当采用正式描述时，NP 难题表示不存在已知算法可在多项式时间内计算对应解；然而一旦求解成功，则可证明在多项式时间内可获得某一对应解。NP 难题较为常见，而 NP 完全问题则可视为其子问题。若存在某一多项式时间算法可对上述两类 NP 问题进行求解，则二者彼此等价。随后，同一算法也适用于求解多项式时间内的其他问题。NP 完全问题是否可在多项式时间内进行求解尚无定论，但大多数机制对此持否定态度。

鉴于相关问题无法实现快速求解，因而无须对此过分担忧。虽然不存在通用算法可计算既定问题的最终结果，但依然可寻找某些算法以提升计算过程的成功几率，即可在搜索空间内提升遍

历速度。回忆一下第 24 章所讨论的 A*算法，虽然在最坏情况下该算法尚不如穷举搜索，但在大多数场合下，A*算法依然体现了自身的快速特征。

搜索过程中的核心内容主要体现在瓶颈问题的处理，在开始阶段，算法基本等同于 alpha-beta 搜索。一类有效的初始方案可通过手工方式求解简单示例，进而查看问题的重点所在。在图 26.1 所示的填字网格中，可于先期随机填入 1ac，即单词 SPACE。随后，下一步将要考察的单词是 2dn，该单词始于字母 A（与 S 和 E 相比，首字母为 A 的单词较少）。接下来则是选择单词 ANVIL，随后的重点则是中间位置处的字母 V。当然，也可放弃 ANVIL 而选择 ALTER，这将获得较多的选择权。

上述过程可通过半正规方式描述为：选择一个单词并针对填字游戏最大化最小选择数量（相信读者对此不会感到陌生），并于随后搜索包含最小选择数量的剩余谜底格。若特定的谜底格被占据（即不存在单词与其匹配），则可回退一步并尝试选择其他单词。

【提示】鉴于当前任务较为简单，因而此处无须进一步查看基于模式匹配的、各数据项的字典搜索。对此，存在大量的程序可执行这一操作，且大多数语言均包含了正则表达式等功能，进而可有效地降低搜索难度。

上述策略始于瓶颈搜索，对应处理过程即为深度优先搜索示例，第 24 章曾对此有所讨论，该方法适用于多种应用场合。总体而言，此类问题涉及多个共存选项的并行选取。

另一个搜索示例则是多格搜索，如图 26.2 所示。多格问题涉及特定空间内的、形状集合的匹配操作。图 26.2 显示了部分示例，其中，重点在于标有“X”的方格，该方格仅能通过某一形状进行填充。除此之外，还可考察十字形组件，并仅可在网格中的 3 个位置进行匹配。

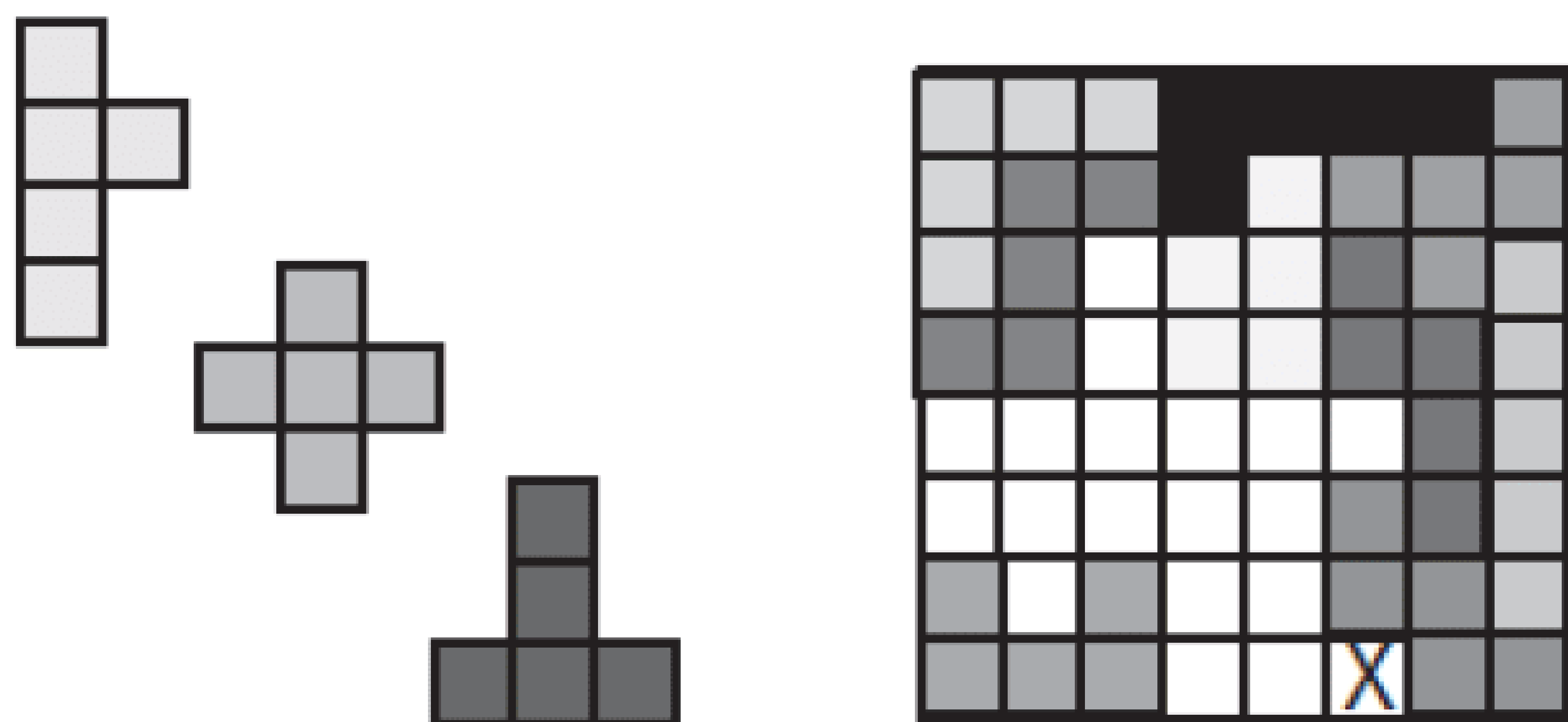


图 26.2　多格问题

宽度优先搜索可视为深度优先搜索的替代方案，并以并行方式考察各阶段中的全部备选方案。通常，鉴于需要考察多种可能情况，宽度优先搜索的效率要低于深度优先搜索。然而，在某些场合下，宽度优先则适用于搜索深度较浅的搜索行为。

不难想象，上述情形较少出现。一般地，可考察某些启发式方案，并优先选取某些求解路径。对此，可将深度优先和宽度优先策略整合为一种更加高级的方法，并结合二者的优点，这与 A*算法十分类似。同时，这形成了全部可能路径的浅层分析，进而对整体考察方案提供路径选择结果。针对基于深度优先搜索的路径选择，此类方案提供了一种更为高级的启发式操作。第 24 章曾对这一类预见性策略有所讨论。

26.2.3　交互行为

这里，重点内容在于如何获取求解问题过程中处理操作的显示结果，以及对该过程的影响方式。若问题求解过程耗时数小时或数天（计算机时间），则通常难以了解当前处理的工作方式。更为重要的是，若计算机在允许过程中崩溃，则全部操作需要重新开始。

一类审核结果可描述为：尽管基于问题求解的处理过程具有一定优势，但作为一般规则，通常不应在搜索算法中使用直接递归操作。此类问题自身即可产生递归结构，因而稍有不便。为了查看相应的工作方式，下面再次考察填字游戏。填字游戏可归结为如下 3 个步骤：

（1）尝试填充下一个谜底格。

（2）若操作无效，则回退并针对前一个谜底格尝试另一个可能的单词（若不存在其他单词且为首个单词，则当前问题无解）。

（3）若填写成功，且为最后一个单词，则任务完成；否则，搜索下一个单词并重复前述步骤。

此类递归结构在深度优先搜索中相对自然，其算法构造过程也较为简单。然而，这将迅速导致调用栈过于庞大，进而导致内存资源紧张。除此之外，若计算机崩溃，则栈中全部内容均会丢失。对此，需要对递归执行回滚（unroll）操作。针对低估结构的保存工作，需要创建自身的调用栈进而对其进行模拟。这使得搜索行为可划分为多个离散步骤，而非运行期内的单一函数调用。

随后，可利用函数的离散步骤并在其间穿插显示当前搜索状态。例如，可生成部分填字游戏。除此之外，还可以文本文件方式（表示当前最优结果）定期显示快照。这里，最优结果包含了各步骤中多种可选方案，并整合了搜索中的多个操作步骤。

26.3　用　例　学　习

下面讨论如何运用前述概念处理复杂问题。对此，尝试编写程序并计算棋类问题。若给定特定的将杀位置，如何搜索到对应的初始位置？例如，白方移动并在 n 处将杀。虽然该问题较为特殊，但却涵盖了诸多重要问题。

在下面的讨论中，假设读者了解些许行棋规则，但却并不深谙赢棋之道。类似地，若一位选手的王被将杀，则另一位棋手获胜。

本小节并未提供相关函数的实现结果，仅定义了相应的函数名以显示对应含义。另外，读者还可尝试引用前述章节中的某些源代码示例。

26.3.1　前期准备

如前所述，此处需要定义一个搜索空间及其表达方式。在当前示例中，该过程较为简单。这里，棋盘表示为 8×8 阵列，对应位置可为空或下列不同颜色（W 或 B）的 6 个棋子之一：兵（P）、马（N）、象（B）、车（R）、王（K）或后（Q）。出于简单考量，棋盘各列采用字母标记，各行

则采用数字标记。当采用这一方案时，A1（或 a1）位于棋盘的左下角，如图 26.3 所示。另外，另一个与棋子相关的信息则是哪一方棋手执行下一步行棋？

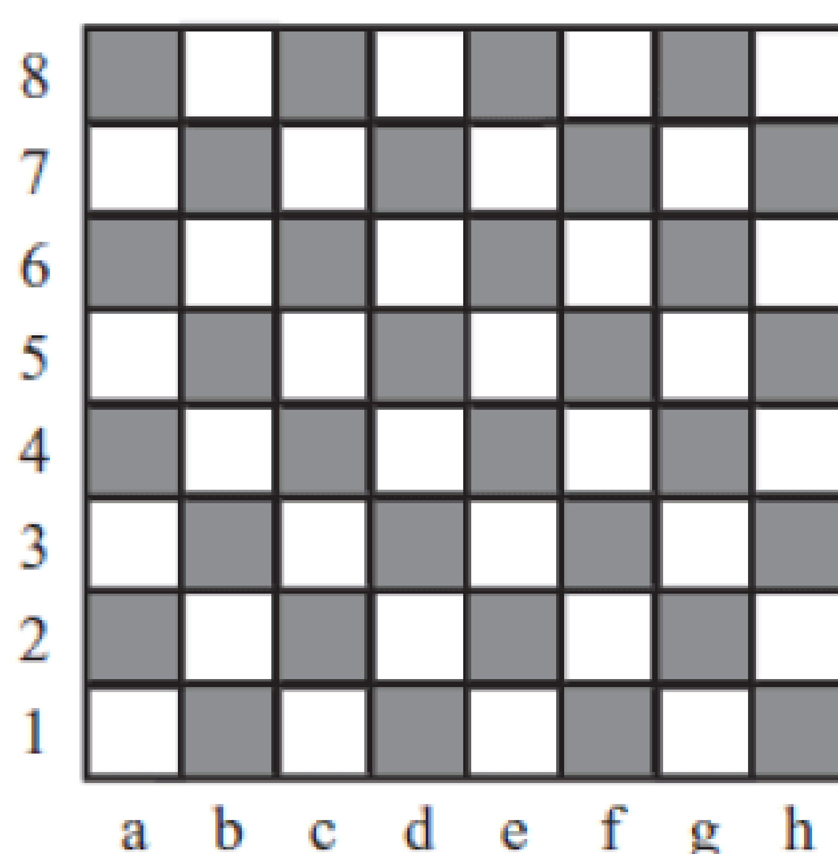


图 26.3　棋盘示意图

对于完全自由的行棋过程，搜索空间的限制条件包括：各种颜色的棋子均包含一个“王”，当然，读者也可定义其他限制条件并应用于任意位置。鉴于某些特殊规则，在一些极端的场合下，大多数限制条件可能难以得到满足。例如“兵”的晋升。下列内容列举了若干限制条件：

- 任意时刻，棋盘上各颜色最多包含 8 个兵、两个马、两个象、两个车以及一个后。
- 若某一颜色包含两个马，则须出现于不同颜色的方格内。
- 兵不得位于行 1 或行 8 中，当该棋子移至行 8 后，其身份发生变化。
- 棋手不可在被将杀时处理其他棋子，也就是说，“王”不应受到威胁。

其中，除了最后一个限制条件之外，其他各项称作语法限制条件。换而言之，读者无须了解行棋规则即可确定是否违背了限定条件。相应地，读者仅需计算棋子，查看盘面状态等内容即可。最后一项条件则称作语义限制条件，为了确认其真实性，读者需了解各棋子的行为方式。

上述限定条件可快速构建初始函数以供搜索使用。其中，possibleMoves(board)函数可视为一个较为基础的函数，该函数将棋盘位置作为输入数据（包括当前棋手的位置），并以有效的格式返回全部移动结果，例如“a1b1”。该函数随后调用 possibleMovesForPiece(board,square)函数，并返回某一特定棋子的全部移动结果。另外，读者还可使用上述函数构造 underThreat(board, square)函数，进而查看移动列表，并以此判断对应棋子置于某一特定方格内。若棋子的移动行为满足上述条件，则目标方格视为处于受威胁状态。

possibleMovesForPiece()函数需要实现相应的行棋规则，其过程并不简单，特别是需要考察某些特殊的行棋步骤，例如车或兵消除“过路兵”。总体而言，此类特殊移动行为较少出现于当前问题中，唯一需要注意的问题是兵的晋升过程。

在上述函数的基础上，可构造 validBoard(board)函数，若棋盘位置有效，则该函数返回 TRUE，否则返回 FALSE。同时，该函数考察前述各种情况，并使用 underThreat()函数确定对方棋是否处于被将杀的状态。除此之外，还可使用 validBoard()函数，并通过 validMoves()函数检测行棋手是否有效，进而改善 possibleMoves()函数。对此，待棋子移动完毕后，validBoard()函数可判断玩家的“王”是否受到威胁，也就是说，可快速确定棋手是否处于将杀状态。这里，“将杀”是指“王”受到威胁，且不再存在有效的行棋步骤；相应地，若“王”无法有效行棋，但却未受到威胁，则“王”处于受困状态。

validMovesInto(board)函数计算全部有效行棋，并生成某一特定位置。由于大多数行棋可逆，因而该函数类似于 validMoves()函数。该函数的复杂之处在于，对于多数行棋而言，移动后的棋子可能吃掉对方的棋子，这也意味着，当还原某一步棋时，棋盘上可能会出现已被吃掉的棋子。

根据上述函数，可定义对应的搜索函数，并可在较好的位置执行该函数，即获取棋盘位置。对应白方将杀，可计算 n 步前的位置，进而满足下列条件：

- 从该位置起，若黑方行棋正确，不存在其他 n 步（或更少）将杀。
- 从该位置起，若白方行棋正确，不存在其他 n 步以使黑方可避免将杀。

26.3.2 编写搜索函数

为了生成搜索函数，可采用大多数易于确认的移动步骤或移动步骤集合。其中，最为简单的例子是“一步将杀”。若给定计算机一个将杀位置，则可利用前述章节所讨论的函数集。具体而言，搜索过程使用 validMovesInto()函数，计算针对该将杀位置的全部棋盘位置，并于随后使用 validMoves()函数获得移动列表。若计算结果不一致，则当前行棋无效——此类问题仅具有一个唯一的正确答案；否则，搜索过程完毕。

这里，应注意上述搜索过程所包含的反复特征。首先，搜索过程回退一步获取预置位置，并于随后从该位置处行棋，此时仅涉及单一移动步骤，进而查看该位置是否适合。其中，各行棋步骤均涉及搜索处理。也就是说，搜索全部可能的预置位置，并获取最为适宜的一步。除此之外，还需搜索全部可能的后置位置（以对当前步骤进行封锁）。为了实现上述两项任务，须制定适宜的启发方案，进而获取最佳搜索顺序。

当确定对应的启发式方案并对搜索过程进行正确引导时，关键之处在于最小化可选方案。例如，当搜索预置位置时，需要针对期望位置优先选择最少的有效移动。在前向搜索中，须对各移动步骤进行检测（与后向移动相比，前向移动的数量较少）。再次强调，须首先搜索较具威胁的移动步骤，这也意味着，使用有效的行棋启发策略，进而相对于期望位置获得最佳步骤。

当处理多步将杀时，情况变得越发复杂，因为此时需要考虑对手的行棋步骤。例如，当出现一步将杀时，需要查看对手的行为方式。该过程涉及后向搜索以及随后的前向搜索。此时，对手被迫移动，因而应对此选取最佳移动。此时，若白方执行下一步后仍处于被将杀状态，则黑方行棋准确有效，任意其他移动（可能也会导致将杀）则为非期望行棋步骤。进一步讲，黑棋可能存在其他行进步骤，但应快于当前目标路径。

当黑棋行棋完毕后，白方需要思考下一步行棋位置，这也意味着另一次反复式检测。无论启发方案如何优秀，依然存在大量内容需要搜索。与简单的遍历相比，该方案旨在快速获取适宜位置。除此之外，该方案还兼具一定的智能性，若读者亲自构建棋类问题，通常也会面临同样的处理过程。

26.4 遗 传 算 法

遗传算法通过计算机模型模拟了自然界的自然选择行为。

26.4.1　自然选择

为了深入理解遗传算法的工作方式，此处有必要考察生命的遗传行为，读者可暂时从数学和物理内容中解脱出来，并转入生物学领域。生物的物理结果主要由 DNA 分子决定，DNA 可表示为链状结构，并由 4 种不同的碱基分子单元（分别采用 A，C，G，T 标记）的副本构成。相应地，一类较为典型的 DNA 链由碱基 AGCCATAGTTACGT 构成。尽管特定生物中的各个细胞包含了具有相同碱基顺序的 DNA 分子副本，但实际顺序也随着物种的不同而产生变化。若非同卵双生（双胞胎）或克隆生物，人类之间的 DNA 通常彼此各异。

在当前环境下，一类有效的方式是将 DNA 链视为包含多个指令的程序，进而制定生物的构造方式。DNA 链由多个独立的基因子序列构成。这里，基因的定义并不严谨且表示为多种结构，包括某种蛋白质的代码序列，以及 DNA 的抽象组成部分，进而确定生物特性。从程序设计角度来看，可将基因视为函数或对象。这里，完整的基因集合称作基因型，而基于基因构造的生物则称作表型。

对于生物而言，最为简单的繁殖方式是采用准确的 DNA 副本产生新生物，该方式称作无性繁殖，大多数细菌、菌类以及植物均采用这一方式繁殖。无性繁殖具有快速的特点，但也会导致某些问题的出现。若全部生物均源自同一基因类型的副本，则此类物种易受到疾病、捕食者以及寄生虫的影响。若某种疾病可感染到一种生物，则全种群均会面临危害。

与无性繁殖相对应的是有性繁殖，这里，两种不同的个体携带不同的基因类型，结合后可产生独特的个体，此类繁殖方式需要 DNA 结构间彼此相容，且双方需要包含沿 DNA 链类似位置的相似基因组（染色体组）。例如，若 Romeo 和 Juliet 计划生育小孩，则决定 Romeo 眼睛颜色的 DNA 对应于 Juliet 眼睛颜色 DNA 的同一位置，即基因的不同版本等位基因（在程序设计中，这类似于同一类的不同实例且包含不同的属性）。

对于绝大部分基因而言，后代继承了两个等位基因（而非一个）且分别来自父母双方。例如，Romeo 和 Juliet 后代的眼睛颜色包含两个基因，其中的一个为显性等位基因，并决定孩子的眼睛颜色。同时，二者皆可遗传至孩子的后代。另外，若染色体组以相同方式加以组织，则对应生物属于同一物种。

有性繁殖的孕育过程相对简单，首先，双方须提供一定数量的生殖细胞——对于哺乳动物而言，即精子和卵子。除了包含各自基因副本之外，精子和卵子与正常细胞相比并无太多不同。这里，关键问题在于有丝分裂，即两个不同基因分别位于细胞中，分裂后生成两个配子。随后，等位基因一分为二并各自进入两个配子中。

等位基因的分布呈随机状态，特定的等位基因究竟源自父亲或母亲则无关紧要，且在配子中的出现几率相同。随后，配子经输送后并查看是否可获取异性的配子。若成功，则将半数基因组（或全部未配对基因组）与另一方配子的基因组结合，进而创建全新的基因组。在该处理过程中，偶尔会存在某些复制误差，即基因突变，这也意味着，后代的等位基因并非与上一代完全保持一致。

在类似物种的种群中，仅部分成员可成功繁殖后代，大量成员在交配前死于疾病或被捕食者消灭；某些成员则因无法吸引配偶而无法繁殖后代。最终，仅相对优秀的基因得以延续。

等位基因的成功延续称作自然选择，这一观点首先由 Charles Darwin（1809—1882 年）提出。这里，“选择”与人为选取以及选择育种过程（农民据此提升农作物以及家畜的产量）有几分类似。自然选择过程出现于多种场合，不难发现，物种可向其后代传递自身特征，种群不同个体间存在些许差异，同时也存在资源竞争等问题。有性繁殖可视为基因传递的良好方式，进而可有效地改善物种传播的成功几率。

在自然选择过程中，生物各代不断演化以适应变化的自然环境。与人为选择不同，自然选择可视为一种盲目过程，演化过程并未指定特定目标，取决于世代繁衍的成功率，该过程仅以渐进增长方式进行。然而，事实表明，鉴于某种进化压力的存在，也会存在某一方向上的直接进化过程。例如，人类在进化压力的作用下，逐渐形成了直立行走、发达的大脑以及使用工具等进化特征。

演化压力可视为一种涌现现象，从该意义上讲，这也是大量微观进化事件的结果。根据前述搜索算法，一种关于进化压力的假设可描述为：生物不断探索自身的生存空间（搜索空间）。当然，这里存在大量的方式可孕育生物。特定的物种通常包含较少的变异，并趋向于搜索空间的同一区域，即进化生态位。对于某些未知区域，物种将获取某种进化优势，搜索空间内的最新发现可供后续探索予以借鉴。若证明该领域确实占优，则物种整体沿该方向进化。

在现有的已知生物中，基于自然选择的进化过程可视为一类低效的行为，毕竟，人工选择具有快速、稳定等特征。但自然选择依然是一种较好的进化方式，并可发现某一问题的隐藏解决方案。当出现较大的进化压力时，生物可在几代内实现动态进化，尤其是有性繁殖。

26.4.2　遗传算法分析

遗传算法将进化方案用于复杂空间的搜索，其行为与生物模型十分接近。鉴于神经元网络可视为生物神经元的理想化模型，因而当前遗传算法同样为理想化方案。

下面再次考察棋类游戏，并讨论 8 皇后问题。为了构造遗传算法，可根据染色体组制定搜索方案。从程序设计角度来看，可用数值列表表达搜索空间，也就是说，使用 64 个数据元素列表，且各数据分别表示为 1 或 0。其中，各 8 位数据不是 8 皇后的对应位置。若存在副本，则该表无效，但依然可忽略无效基因类型进而对该表进行处理。此类基因类型可视为非活性遗传性状。

算法构造过程中的其他重要因素还包括工具函数，并告知与正确方案之间的接近程度。对此，可计算各皇后处于威胁状态下的总数量。如图 26.4 所示，对应函数计算至 8。当前目标是最小化工具函数的返回值，针对各种生物，对应函数的返回值称作适应度景观（fitness landscape），并表示为不同答案的有效表现方式。从图形上看，景观谷底表示有效区域，读者应尝试针对最低值搜索谷底区域（或为 0 值的谷底）。

工具函数在遗传算法中表示为一类最弱的链接，若当前问题为一类极端情形（或者全有，或者全无），则不存在有利条件以接近正确处理方案，且算法难以获得有效的结果。相应地，适应度景观中的谷底包含了陡峭边，因而算法无法对其进行计算。该问题可视为神创论者的经典反驳观点，即中间形式问题。

自然选择学说的提倡者认为，若生物结构在进化各阶段不存在直接优势，则是进化论中的一个严重问题。从计算角度上看，通常可获得相应的根据函数，并可得到特定方案的有效结果。而

且，此类函数通常为粗粒度型，并包含大量的、相等适应度的不同表型——实际上，这可视为算法的优点。

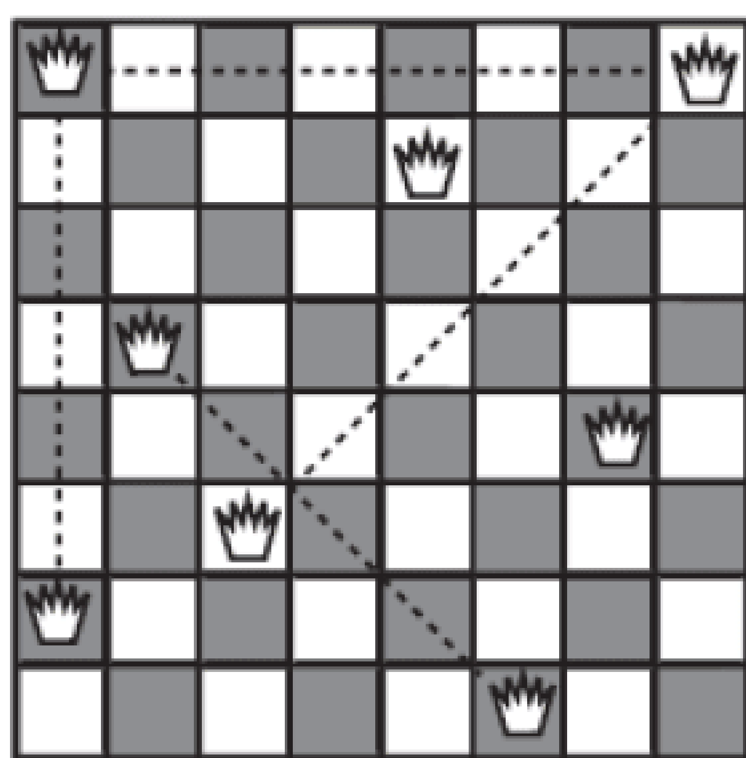


图 26.4　8 皇后问题及其“基因”型

遗传算法的基本原理将使用到小族群计算型生物，并通过工具函数对其进行计算。其中，仅最优（包含最低分值）生物可实现“繁殖”行为，进而获得下一代物种。这里，“繁殖”过程采取有丝分裂方式进行，即使用各生物的 DNA 链并对其进行接合，splice()函数封装了上述行为，如下所示：

```
function mateOrganisms(strand1, strand2)
  set child to an empty array
  set currentStrand to strand1
  set length to the number of elements of strand1
  repeat for i=1 to length
    //randomly switch strands
    if random(length)<5 then switch currentStrand
    set element to currentStrand[i]
    //randomly mutate the occasional element
    if random(length)=1 then set element to not(element)
    add element to child
  end repeat
  return child
end function
```

尽管 mateOrganisms()函数采用任意方式设置，但依然存在大约一次基因突变，且每个 DNA 链的 5 次转换均工作良好。虽然该过程充满了奇幻色彩，但当前算法依然趋于最优方案。而且，算法较好地处理了适应度景观中的局部最小值问题。也就是说，算法接近于最优结果，但却无法实现进一步扩展。另外，谷底最低点并非真正的最小值。对于全部搜索策略而言，局部最小值依然是最大问题。然而，遗传算法中的随机元素使得相关操作可探索搜索空间内更为广泛的区域，并于同时关注当前主要路线。

26.4.3　调整过程

当前系统体现了一定的简化特征，但依然存在多种方式可调整遗传算法，并以此改善搜索率，这主要取决于局部最小值的数量。遗传算法的一个主要问题是计算过程趋于不变性，即基因库缺

乏应有的变化，并仅呈现一类基本模式。相反，过量的突变则意味着，算法在转换前（通过替换方案）无法获得有效解。下列内容列举了某些转换方法：

（1）在标准算法中，进化获胜者之间并无明显区别。在各个阶段，该领域仅简单地一分为二。这里，假设存在 15 个生物并选取 6 个最优个体与剩余成员逐一匹配，进而产生 15 个新成员。一种替换方案可描述为，可对优生、多育的个体予以“奖励”，进而对操作结果执行加权计算。此处，获胜者的匹配率加倍，而失败者将不再参与繁殖过程，这将显著提升问题的求解过程，并包含少量或不包含任何局部最小值。

（2）读者可尝试使用辐射变量，这将对突变率和交叉率产生影响。通过定期增加或降低辐射变量，当关注于当前路径时，可生成稳定的算法周期，并穿插于某些不稳定期（其中，可对更多的广布理念进行测试）。本书源代码中的 geneticAlgorithm()函数即为一例，这将有助于消除不变性问题。但该过程应保留某些有效的基因类型，以防止其意外丢失。

（3）读者可适当调整种群大小，并维持更加灵活可变的基因库。该方案旨在降低差异性，通常情况下，包含 50～100 个成员的族群往往较优。

（4）总体而言，新一代个体可完全取代上一代个体。相反，可尝试令上一代个体与其后代共同参与竞争，以使上一代成员中的最优特征不至于完全丧失。

（5）这里，可通过消除大量的种群以使其消亡，其中可能包含较优的物种。例如，传染病可对种群中较为普遍的基因类型产生破坏，并使劣势者可安然渡过这一危险期。

（6）可使某类物种脱离于当前种群并独自进化。随后，可将其再次融入种群，并参与竞争和繁殖。

（7）可引入成对基因系统，其中，各生物继承两个等位基因而非一个。然而，该过程需要某些合理方法以使某类等位基因占优。

上述大多数转换方法旨在模拟生物学条件，特别是人工选择过程。另外，还可针对自然选择提供相应的辅助条件，进而加速进化过程。然而，全部搜索策略中亦存在一定的危险性，即算法的难度过大，进而丧失其固有的优点。

对此，一类替代方法将调整工具函数，并可通过修改操作粒度予以实现。如前所述，粗粒度工具函数可有助于消除局部最小值。相应地，细粒度操作可有效地简化计算过程。

此处，可使用第二个工具函数，并通过替代方案计算对应生物的成功率。例如，当对微分方程求解代数方案时，相应的测算过程可能分别包含“简化”过程和“精确”过程。理想化的生物对象可对二者进行最小化操作，但通常情况下，二者间存在一类折中方案。当两个生物之间进行比较时，若某一生物可较好地处理上述两种测算过程，则该生物视为占优；相反，若无法较好地处理两种测算过程，则该生物同样优异。

由于需要在各阶段计算种群中的各成员，因而工具函数可视为算法中计算量较大的部分。加速该函数的计算过程可在特定时间内产生更多的世代。

26.5 本 章 练 习

【练习 26.1】 根据本章所述内容，尝试构造遗传算法并解决 8 皇后问题。其中，可在棋盘

上放置 8 个皇后棋子且彼此均不受威胁。

26.6　本 章 小 结

本章通过多种情形讲解了相关搜索方法，进而对某些较为困难的问题进行求解，并在讨论前述 AI 扩展问题的同时对遗传算法予以分析。

本书第一部分内容阐述了简单的数值原理以及基本的代数运算，根据基础的数学和物理学知识，读者可了解复杂技术的构造及其于程序设计的应用方式，特别是游戏领域。尽管本书对某些难度较大的话题仅做了简要介绍，但读者可借此考察相关问题，进而了解某些专业术语。同时，在实际应用过程中，读者可力争做到有的放矢。更为重要的是，读者还可在原有内容的基础上进行创新，例如，读者可尝试构造 AI 机器人或者编写 3D 保龄球游戏。

至此，读者应掌握如下内容：

- 如何应用搜索空间对问题进行分类。
- 如何采用深度优先以及宽度优先策略遍历搜索空间。
- 遗传算法和进化生物学之间的关系。
- 如何构造染色体基因以表达搜索空间。
- 工具函数以及适应度景观的含义，及其遗传算法的应用方法。
- 遗传算法的速度优化策略。

附录A　术　语　表

尽管并未涵盖全部数学内容，但该术语表概括了基本的相关术语。由于某些术语的含义较为明显（例如点、形状等），因而这里未涉及其复杂、严格的定义。针对书中未出现的其他术语，此处将给予简要的解释，尽管大多数定义缺乏严格的数学推导过程。

绝对值——对于实数 n，若 $n < 0$，则其绝度值表示为$-n$；否则为 n。换而言之，除了 0 之外，数字的绝度值均为正值。

加速度——速度或速率的变化率。

锐角——小于直角的角度。

仿射转换——保持平行线的转换操作。

空气阻力——在空气中运动对象受到的作用力，并对运动状态产生抑制作用，也称作流体阻力或阻力。

代数解——针对公式中其他变量的通项，相对于数值解，代数解表示为未知值的计算结果。

算法——包含多个预定义计算的处理过程，该过程接收特定的参数（实际上可表示为包含输入数据的程序）。

锯齿——直线向量转换为硬边像素时产生的锯齿状图案，读者可参考术语“抗锯齿”。

环境光——作为光线在各个方向上的反射结果，环境光表示为某一区域的全方位照明。读者还可参考术语“有向光源”和“衰减光源”。

振幅——在振荡过程中，相对于平衡位置，振荡器产生的最大距离。

角——旋转过程中的量值，定义为圆的某一部分，通常采用角度或弧度表示。

入射角——对象或波与某一表面碰撞时的角度。

反射角——碰撞完毕后，对象或波与某一表面间的反弹角度。

角频率——在某一特定时间段内，旋转对象所完成的旋转量（参见术语“频率”）。

抗锯齿——通过在颜色和背景间执行渐进式插值，直线向量可转换为平滑的彩色像素。

参数——向数学函数或程序函数提供的输入变量。

支架——针对碰撞检测，该术语是指用于简化对象描述的多边形或多面体。另外，读者还可参考术语“碰撞图”。

结合律——针对定义域内的数字 a，b，c 和操作符#，$a\#(b\#c)=(a\#b)\#c$（例如 $a+(b+c)=(a+b)+c$）。另外，读者还可参考术语“交换律”和“分配律”。

渐近线——函数值无限接近的直线或平面。

衰减光源——源自某一特定点的光源，光照在传播过程中随距离而减弱。另外，读者还可参考“环境光源”和“有向光源”。

平均值——数值集中间值的统称，也称作均值。

轴——在笛卡儿空间内，直线穿越原点并平行于某一基向量。

旋转轴——对象旋转所围绕的直线。

烘焙纹理——光照预先计算后的纹理贴图。

弹道学——当仅受到重力作用时，研究对象运动方式的一门学科。

质心坐标——在齐次坐标系内，点 P 的坐标通过 3 个三角形的有符号面积进行计算。其中，对应三角形由点 P 和预定义点 A，B，C 构成。

基向量——向量集合和既定原点，用于确定笛卡儿空间（以及其他向量空间）。

Bezier 曲线——连续控制点之间 3 次函数所定义的样条，且在各控制点处包含所定义的切线。另外，读者还可参考术语“Catmull-Rom 样条”。

布尔代数——与布尔数协同工作的系统，相关数字仅包含两种可能值：TRUE 和 FALSE，并使用 AND，OR 和 NOT 操作符进行计算。

包围体——对于 3D 空间内的形状，完全包围该形状的体空间，类似于“包围区域”。

包围体层次结构——3D 空间内的划分树（类似于 2D 空间中的“包围区域层次结构”），其中，对象被连续划分为较小的区域（并包含较小的数据集）。

交叉法——该方法用于细化求解区间，以及计算近似数值解。

宽度优先搜索——该方法通过检测特定级别上的各个分支以遍历搜索树。另外，读者还可参考术语“深度优先搜索”。

凹凸贴图——用于描述特定点表面高度的纹理贴图。

微积分学——研究无穷小值的一门学科，特指积分和微分。

笛卡儿坐标——根据基向量，用于在笛卡儿空间内定义点位置的数值集。

笛卡儿平面——二维笛卡儿空间。

笛卡儿空间——向量（包含 n 个实数）定义的点集，例如多个点形成的三角形其内角和为 180°。

Catmull-Rom 样条——在连续控制点间，3 次函数定义的样条，其中，各曲线段由 4 个控制点加以定义，进而生成平滑的过渡。另外，读者还可参考术语“Bezier 曲线”。

质心——对于任意对象，任意直线或平面穿越某一点，并均分两侧的质量。

离心力——圆周运动对象产生的作用力。另外，读者还可参考术语“向心力”。

向心力——保持对象圆周运动的作用力。另外，读者还可参考术语“离心力”。

中心——三角形的质量中心。

子节点——在树形结构中，当且仅当 y 表示为 x 的父节点时，节点 x 称作子节点。

圆——相对于既定平面内的另一点，圆定义为包含固定距离 r 的点迹。

圆周——圆的周长。

系数——表达式中某一项的常量部分。例如，$4x$ 项的系数表示为 4。

弹性系数——弹簧拉伸长度与张力间的比例常数。

摩擦系数——垂直于表面接触面的作用力与（抑制对象运动的）摩擦力之间的比例常数 μ。

共线——在 3 个或更多点中，各点均位于一条直线上。

碰撞图——针对碰撞检测，以简化方式描述对象形状的图像贴图。

公因子——在数字 a 和 b 中，某一数字（或表达式）为 a 和 b 的因子。

交换律——针对操作符#和定义域内的 a 和 b，$a \# b = b \# a$（例如 $a+b=b+a$）。另外，读者还可参考术语“结合律”和“分配律”。

复数——实数与虚数之和这一形式的数字集，通常记为 C。

计算复杂度——算法运行时的时间量度，并可视为参数尺寸的函数。

凹面——即非凸面形状。

圆锥体——直线围绕空间内非平行直线的旋转表面。另外，读者还可参考术语“圆柱体”。

同余——针对某一整数 n，当且仅当 $a=b+nm$，整数 a 和 b 同余基数 m。

共轭——在复数或四元数中，虚部互为相反数。

能量守恒定律——该定律表明，若对象集不存在合力（不同于势能计算），则全部能量系统保持恒定。

动量守恒定律——该定律表明，对于不存在外力的粒子系统，该系统的全部动量保持恒定。

常数——当进行函数计算时，函数中视为固定不变的数据元素。

比例常数——参见术语“比例项”。

控制点——在样条中，以某种方式定义曲线的数据点，通常为该曲线上的点。

凸面——在某一形状中，周长上任意两点间的直线均不会位于该形状的外部。另外，读者还可参考术语“凹面”。

坐标——参见术语“笛卡儿坐标”。

互质——当且仅当两个自然数 a 和 b 的最大公约数为 1 时，二者互质。

可数集——在某一集合中，其数据元素可通过某种方式列出，即使对应列表永久持续（当采用自然数时，对应数据以一一对应方式设置）。

可数数——参见术语“自然数”。

耦合振子——连接在一起的两个或多个振子，且彼此施加作用力。

临界值——若超出该值，则系统行为将产生性质上的变化。例如，若光线大于临界角，则折射时无法脱离于某一介质。

叉积——参见术语“向量积”。

三次多项式——阶数为 3 的多项式。

曲线——由独立连续变化参数定义的、空间内的连接点集。

圆柱体——直线围绕另一平行直线的旋转表面。另外，读者还可参考术语“圆锥体”。

阻尼简谐运动——振子的简谐运动，通常包含一个阻尼因子。

阻尼——与振荡行为反向的因子，且正比于速度。

阶数——多项式函数 $f(x)$ 中 x 的最大指数。例如，函数 x^3+2x 的阶数为 3。

度——角度单位，定义为圆的 1/360。

分母——分数中的下方数字（也称作除数）。

深度优先搜索——通过检测特定分支直至成功或失败，并于随后执行回溯操作的搜索树遍历方法。另外，读者还可参考术语“宽度优先搜索”。

导数——函数 g 针对函数 f 的参数值生成梯度。读者还可参考术语“积分”和“变化率”。

行列式——矩阵的“量值”，当作为转换操作时，矩阵根据该量值缩放一个立方体对象。

对角线——连接某一形状两个顶点的直线。特别地，对角线通常是此类直线中的最长直线。

差——两个数值相减后的结果。

微分方程——函数及其导数关联的方程。

微分——计算函数梯度（导数）的过程。

衍射——波途径狭小缝隙时将改变其形状，该缝隙貌似为新波源。

漫反射——针对表面的入射光，无特定方向的反射行为，也称作 Lambertian 反射。另外，读者还可参考术语“镜面反射”。

维度——针对某一向量或笛卡儿空间，须定义的分量或基向量的数量。

有向光源——发射自特定方向上的光源，且光线不随距离的变化衰减（例如阳光）。另外，读者还可参考术语“环境光”和“衰减光源”。

判别式——判别式最终值源自函数的参数，其值可通过某种方式划分函数的行为，例如计算根值。

位移——两点间的对应向量。

距离——两点间直线的长度，即二者间向量的大小。

分配律——针对操作符@和#，以及定义域内的 a，b，c，有 $a \# (b @ c) = (a \# b) @ (a \# c)$，例如 $a\times(b + c) = (a\times b) + (b\times c)$。另外，读者还可参考术语“交换律”和“结合律”。

除数——参见术语“因子”。

定义域——函数所定义的数值集合。

多普勒频移——波长在不同的参考坐标系中具有不同的频率。

点积——参见术语“标量积”。

阻力——参见术语“空气阻力”。

e——该值表示为 2.718…，函数 $x\to e^x$ 的导数等于其自身。

离心率——椭圆偏离圆的量值，并通过 $\sqrt{1-\frac{b^2}{a^2}}$ 加以定义。其中，a 和 b 分别表示为最大半轴和最小半轴。

效率——系统中可转化为有用功的能量比值。

弹性极值——弹簧保持有效弹性系数的最大拉伸长度。

弹性势能——对象在拉伸弹簧或弹力作用下的势能。

椭圆——平面内点的轨迹，其距离两个定义点的距离等于常量。

椭球体——应用于球体对象上的仿射结果。

突发现象——简单交互处理的群集行为。

能量——对象运动或处于运动势态时包含的量值。读者可参考术语“动能”、“势能”、“重力势能”、“弹性势能”以及“能量守恒定律”。

方程——两个数值彼此相等且通常包含一个或多个未知项。

等边三角形——3 条边等长的 2D 三角形。

平衡状态——不存在合力时对象系统的状态。

估算函数——此类函数负责计算问题特定解与正确答案之间的接近程度。

欧几里德空间——参见术语“笛卡儿空间”。

指数——数字相乘的幂值。例如，在 x^5 中，x 的指数为 5。

指数函数——此类函数的行为类似于幂函数映射 $x\to e^x$。另外，读者还可参考术语“对数”。

表达式——包含数据项组合的函数，例如 $3y^2+ 10xz -5(x^3 - 2)$。

伸展长度——弹簧或其他弹性材质相对于自然位置的伸展量。另外，读者还可参考术语“弹性极值”。

因子——针对数字 a 和 b，当且仅当 b 为 a 的整数倍数时，a 为 b 的因子。

因式分解——将函数或数值划分为多个因子的过程。

视域——距相机特定距离处的可见范围；或者相机视口形成的对角。

有限集——包含有限数量数据元素的集合。

焦点——椭圆中由两个数据点之一加以定义。

公式——包含多个变量的方程，且数值间彼此关联（通常将某一变量定义为其他变量的函数）。公式通常不包含未知项，并假设对于全部可能的变量值其结果均为真。

商——两个数字间相除后的结果（当两个数均为整数，对应结果为有理数）。

频率——振子在特定时间内完成全振荡的次数。另外，读者还可参考术语“周期”。

摩擦力——平行于运行方向、两个表面间产生的作用力，通常抑制对象的运动状态，且正比于垂直于表面的作用力。另外，读者还可参考术语“摩擦系数”。

视锥体——截取金字塔状对象或圆锥体顶端后的 3D 形状。

支点——杠杆或其他对象围绕旋转的点或直线，即旋转轴。

函数——一个数据集（定义域）和另一个数据集（值域）之间的映射关系。函数可通过代数方式描述（例如 $f{:}x\to x^3$，或简写为 $f(x)=x^3$），或者简单地加以描述。例如，若 n 为奇数，则 $f(n)=1$，否则 $f(n)=-1$。

遗传算法——一类模拟自然选择并通过“演化”方案在搜索空间遍历的方法。

全局最大值——基于任意参数值的函数最大值（类似还存在全局最小值这一概念）。另外，读者还可参考术语“局部最大值”。

梯度——直线在笛卡儿空间内的斜率，其定义方式可描述为：垂直移动距离除以水平方向上的无穷小移动距离。

图——基于参数的结果值所绘制的函数图，或者由边连接的节点集。

重力势能——对象在重力作用下的势能。

最大公约数——针对两个自然数 a 和 b，二者公因子中的最大值。

贪婪算法——该算法优先搜索问题的局部区域解。

半长轴——在椭圆中，位于周长上两个最大极值点之间一半的距离（沿穿越两个焦点的直线）。相应地，半短轴则是该直线距周长之间的最大距离。

齐次坐标——通过 $n+1$ 维中的直线投影表达 n 维空间内的一点。

斜边——在直角三角形中与直角相对的边。

单位元素——该值经某种运算符计算后使得其他值保持不变。例如恒等函数 $f(x)=x$，主对角线为 1、其他数值为 0 的单位矩阵，加法运算中的 0 以及乘法运算中的 1。

当且仅当——在并且仅仅在其他条件成立时，某一命题方成立，即二者皆为真或皆为假（该术语有时也记为 iff）。

图像贴图——通过某种映射方法，将（2D 环境中的）图像投影至某一表面上，反之亦然，并以此展现表面上的相关属性。另外，读者还可参考术语“碰撞贴图”、“纹理贴图”和“凹凸贴图”。

虚数——若−1 的平方根定义为值 i，则该值的任意倍数均称作虚数。从某种意义上讲，虚数集垂直于实数集。

独立方程组——两个或多个联立方程且彼此无关。

不等式——两个数值通过某种方式彼此关联（例如某一值大于另一值）。另外，读者还可参考术语“等式”。

惯性——参见术语“质量”。

初始条件——系统的初始状态可推断出其后续行为，当使用微分方程时尤其如此。

整数倍数——针对两个数值 a 和 b（数字或表达式），当且仅当存在整数 n 且满足 $b = an$ 时，b 表示为 a 的整数倍。另外，读者还可参考术语“因子”以作比较。

整数——自然数和负自然数构成的集合（…，−2，−1，0，1，2，…），记为符号 $\mathbb{Z}$ 。

积分——函数 f 在另一函数和轴向间生成无穷小的有符号切片面积（不定积分）；或者通过代入两个参数值，计算两点间曲线下方的面积（定积分）。积分的逆运算为导数计算。

积分法——计算函数图和水平轴之间的面积。

截距——对于函数 $f(x)$， $f(0)$ 对应的值。

插值——通过参数化两个不同值之间的直线，计算其间的中间值。

区间——实数集合包含开放区间（不包含端点，例如集合$-1 < x < 0$）、闭合区间（包含端点，例如集合$-1 \leqslant x \leqslant 0$）以及二者的混合结果。

反函数——针对一一映射函数f，其反函数记为 f^{-1}，并在f的值域内加以定义，即若 $f(x) = y$，则 $f^{-1}(y) = x$ 。同样，针对f定义域内的全部x，有 $f^{-1}(f(x)) = x$ 。

逆向动力学（IK）——计算一系列连接刚体的运动行为，进而实现特定目标。

逆矩阵——对于包含非 0 行列式的矩阵 $\mathbf{M}$，矩阵 $\mathbf{M}^{-1}$ 满足 $\mathbf{M}^{-1}\mathbf{M} = \mathbf{M}\,\mathbf{M}^{-1} = \mathbf{I}$。其中，$\mathbf{I}$ 表示相应尺寸的单位矩阵。

反比——若存在某一常数 k 且针对全部有效的 x 和 y 满足 $x = \dfrac{k}{y}$，则 x 和 y 呈反比。另外，读者还可参考术语“正比”。

平方反比定律——针对与 x 和 y 相关的公式，x 反比于 y 的平方。

无理数——隶属于实数集，例如 π 和 2 的平方根皆为无理数。

等腰三角形——三角形中的两条边等长。

迭代函数——算法连续使用各后续步骤的计算结果。另外，读者还可参考术语“递归函数”并与此进行比较。

动能——物体运动时具有的能量，定义为$\dfrac{1}{2}mv^2$，这与角速度的计算有些类似。

动摩擦力——两个彼此相对运动的表面间的摩擦力。另外，读者还可参考术语“静摩擦力”。

Lambertian 反射——参见术语“漫反射”。

薄片——三维空间内的一类理想对象，其厚度为 0。

定律——也称作物理定律，通过公式方式阐述物理世界某一类假想的基本真理。

主对角线——在矩阵 m_{ij} 中，左上方至右下方对角线上的值，对应值记为 m_{ii}。

叶节点——在树形结构中，不包含子节点的节点。

直线段——对于向量 **a** 和 **v** 以及值 $0 \leqslant t \leqslant 1$，包含向量 **a**+$t$**v** 的点曲线。

线性函数——阶数为 1 的多项式。

局部最大值——基于参数值的函数 f 值大于近参数的全部 f 值（类似地还有“局部最小值”）。另外，读者还可参考术语“全局最大值”。

轨迹——包含特定属性的点集（例如，直线即可视为平面内的点集）。

对数——幂函数的逆函数。针对数字 a，b 和 x，若满足 $b^x = a$，则 b 称作底数。

最小公倍数——对于两个自然数 a 和 b，二者整数倍的最小值。

最简分数——分数的分子和分母互质。

量值——根据毕达哥拉斯定理，参数和平方根所确定的向量长度。

多对一函数——定义域内多个数据元素和映射为值域中的同一值。另外，读者还可参考术语“一对一函数”和“多值函数”。

映射——参见术语“函数”。

质量——物质的一种属性，用于定义加速时所需的作用力，有时也称作“惯性”。

矩阵——实数的二维阵列。

最大值——函数图上的一点且位于最高端。针对平滑曲线，对应梯度为 0。另外，读者还可参考术语“局部最大值”和“全局最大值”。

均值——对于 n 个数值，其和除以 n 后的计算结果。另外，读者还可参考术语“平均值”。

力学——研究物体运动状态或平衡状态的一门学科。

网格——3D 场景中计算机生成的一种形状，包含了由多边形连接的多个顶点。

最小-最大算法——在游戏中，玩家的运动行为最小化其最大损失（即最大化最小收益），算法利用这一概念计算最佳策略。

纹理链——多种不同尺寸的纹理贴图，用于在不同距离处生成平滑的着色结果。

模型——在 3D 场景世界中，由计算机生成的网格形状。

求模——针对自然数 m，通过除以 m 并将各数值映射为其余数，函数可将全部整数集映射为有限集$\{0, 1, \cdots, m-1\}$，该术语也称作同余关系。

转动惯量——惯性的角运动等价概念，其值等于质量和（与旋转轴间的）距离平方的乘积。

动量——对象的质量和速度的乘积。另外，读者还可参考术语“动量守恒”。

多值函数——在函数中，定义域中的某些数据元素可映射为值域中的多个数据（非严格定义）。另外，读者还可参考术语“一对一函数”、“多对一”函数。

自然数——计数集合且包括 0（0, 1, 2, …），通常记为 N。

网络——图中各边均赋予相关值，以表明某种“开销”。

节点——虚拟对象，图或 3D 空间元素中的顶点。

法线——在两个向量中，二者彼此垂直；在表面上，法线向量垂直于该表面。

标准化——向量除以自身长度得到的单位向量。

分子——分数中的上方数字，也称作被除数。

数值解——通过搜索特定数值以使方程成立，进而计算其中的未知项（代数解的“反义词”）。

钝角——大于 90° 且小于直线的角度。

八叉树——参见术语“四叉树”。

一一对应函数——在函数中，对应于定义域中的数据元素映射为值域中的单一元素，反之亦然。另外，读者还可参考术语“多对一函数”和“多值函数”。

操作符——某一函数将一个或多个参数从特定集合映射为同一集合中的单一结果，例如二元操作符+，–，×，÷。

原点——基向量的起始点。

正交——两个或多个向量彼此垂直。

正交基向量——基向量处于正交状态且长度值为 1。

振荡行为——一段时间内的重复运动，例如弹簧或水波。另外，读者还可参考术语“频率”、“周期”和“振幅”。

异相——两个振荡行为包含相同的频率和不同的相位（例如，相差半个周期）。

成对乘法——在两个向量中，根据各参数分量的乘积并在操作符的作用下生成一个新向量。例如，向量(2 3)和(1 –2)的成对乘法计算结果为向量(2 –6)。

抛物线——二维曲线，表示为二次函数图。

平行——两个向量的标量积为 1；或者，在两条直线中，其上的两个向量彼此平行。

平行四边形——包含两组平行边的四边形。

参数——函数中的数据元素，不同的参数可生成包含类似属性的不同函数。

参数化——根据参数方程定义曲线或表面。

参数方程——变量与真值参数相关的一组公式。

父节点——在树形结构中，针对特定节点，其下一个节点朝向根节点。另外，读者还可参考术语“子节点”。

偏导数——绘制于某一表面上的曲线的导数。

粒子——空间内的一类理想化对象，其尺寸为 0 但却包含质量。通常情况下，粒子对象还包含其他属性。

特解——在给定特定的初始条件后，基于微分方程的函数为真。另外，读者还可参考术语“通解”。

划分树——树形结构根据相对位置描述一组空间中的对象。

周长——定义外部边界的曲线或表面。

周期——振子完成一次振荡所需的时间。另外，读者还可参考术语“频率”。

垂直——两个向量呈 90° 角。另外，读者还可参考术语“正交”。

扰动——利用较小量值调整某一数据值。

相位——包含相同波形的两个波，波前于特定时刻分离时的距离。

平面——距两个定义点相同距离处的、空间内的点的轨迹。也就是说，平面包含位置向量 $\mathbf{a} + t\mathbf{v} + s\mathbf{w}$，其中，**a**，**v** 和 **w** 为定义的向量，t 和 s 为参数。

定期运动（ply）——由玩家所控制的运动。

拐点——函数图中的一点，其二阶导数为 0。

多边形——由连接相同数量顶点的多条直线段构成的封闭形状。

多面体——封闭的 3D 形状，其表面由直边连接的多个多边形构成。

多项式——形如 ax^n 构成的单变量函数，例如 $2x^3+3x^2-4$。

位置向量——原点至特定点之间的向量。

势能——外部作用力影响下的、对象所具有的能量，特别是重力、拉伸弹簧以及磁场等，一般等价于系统所做的功。

幂——某一数值自身相乘的次数。例如，8 可表示为 2 的 3 次方，记为 2^3。除此之外，还存在幂函数这一概念，并可将数值对映射为实数或复数。

功——系统或机器在一段时间内所做的功（释放或使用的能量）。

质因子——对于两个自然数 a 和 b，表示为公共因子的质数。

相对论——全部物理计算应不依赖于计算所在的参考坐标系。

积——两个数值相乘的结果。

抛体运动——物体在重力作用下的运动行为。

投影——降低空间维度的映射过程。例如，球体可映射为圆。

正比——若针对全部有效值 x 和 y，存在比例常数 k，并满足 $x = ky$，则称 $x = ky$ 呈正比。另外，读者还可参考术语“反比”。

象限——笛卡儿平面内的某一区域，其中，全部数据点具有相同的坐标符号。

二次函数——阶数为 2 的多项式。

四边形——包含 4 条边的多边形。

四叉树——3D 空间内的划分树，其中，对象根据其位置被划分为连续的正方形（3D 空间内的对应结构为八叉树）。

四元数——一类特定的 4D 向量，此类向量等价于复数。

商——两个数字间的除法计算结果（参见术语“分数”），或者该商的整数部分（参见术语“求模”）。

弧度——角度单位。

半径——圆周上的点与圆心间的距离。

小数点——在列数字标记法中，小数点表示为数字整数部分的结束位置。

值域——函数所返回的数值集合。

变化率——数值在一段时间内的变化量，等价于基于时间的导数。

有理数——表示为两个整数分数形式的数字集，记为 Q。

射线——从既定点至无穷远处的 3D 空间内的直线。

实数——可在数轴表示的数字集，记为 $\mathbb{R}$ 。

倒数——数字 n 的分数形式，即 $\frac{1}{n}$ 。

直线形式——在直线系统中，两条直线呈平行状态或垂直状态。

递归函数——递归算法的工作方式可描述为：根据基于同一算法的简单情形计算各个步骤，直至获得可直接求解问题的用例。另外，读者还可参考术语“迭代函数”。

反证法——为了证明某一事物正确，可假设该事物错误并引出矛盾结论。

参考坐标系——在物理模拟过程中，参考坐标系指物理空间基向量，以及用于定义其他计算的参考速度。另外，读者还可参考术语“相对论”。

反射——在镜像直线或平面中，各点至对立点间的移动转换。

优角——大于 180° 且小于 360° 的角。

折射——一类物理现象，其中，作为速度变化结果，波将改变其行进方向。

余数——当自然数 n 和 m 相除时，针对自然数 a，值 r 满足 $0 \leqslant r \leqslant m$ 和 $n = am + r$。

共振——作为同一频率的振荡作用力，振荡过程逐渐增加其振幅。

菱形——各边等长的平行四边形。

直角——90° 角。

直角三角形——包含直角的三角形。

刚体转换——保持直线间夹角的转换。

根——在函数 f 中的 x 值，并满足 $f(x)=0$ 。

根节点——在树形结构中，不包含父节点的节点。

旋转——围绕某一特定中心位置，使各点移动某一角度的转换操作。

鞍点——在某一表面中，一点在一个方向上表示为局部最大值，而在另一方向上为局部最小值。

标量——非向量数据，即不包含方向；换而言之，标量也可视为维度为 1 的向量。

标量积——两个向量分量的配对乘积结果之和，记为 $\mathbf{u} \cdot \mathbf{v}$。例如，向量(2 3)和(1 –2)的标量积表示为 2×1 + 3×(–2)（也称作点积）。

缩放操作——相对于某一参考点，缩放转换可描述为：将各点移至自身向量的某一倍数处。

搜索空间——某一问题的潜在解集。

搜索树——表达游戏中全部可能移动的树形结构，并对此寻找相关算法。

集合——对象（或元素）集，包括有限集（例如集合{1,3,5,7}）、可数集（奇数集合）以及不可数集（例如 0 和 1 之间的全部数字集合）。

剪切转换——通过向量倍数且平行于参考直线或平面的方式，该转换将各点按比例地移至距该直线或平面的某一距离处。

符号——某一数值的正负性（例如，点与直线间的距离在某一方向上为正值，而在另一方向上为负值）。

相似性——在两个形状之间，除了缩放转换之外，二者彼此等价。

简谐振动——正弦运动，通常与拉伸弹簧的运动相关。另外，读者还可参考术语“阻尼谐振动”以作比较。

简化计算——通过代数方式处理某一函数或陈述，以使其表现为易处理之形式。

联立方程——相同未知项中的两个或多个方程，且针对相同解集方程均为真。

正弦曲线——形如 $A\sin(\mathrm{w}x + c)$的函数。其中，A 表示为振幅，w 表示为频率，c 表示为相位。

镜面反射——光线直接从对象表面反射，且入射角等于反射角。另外，读者还可参考术语“漫反射”以作比较。

速度——运动对象在一段时间内移动的距离。

球体——与特定中心保持相同距离的空间内的点集（通常为 3D 空间）。

样条——由某一参数函数定义的曲线。

弹簧——可伸展对象并产生张力。同时，弹簧还可被适当压缩。另外，读者还可参考术语“伸展”和“弹性系数”。

正方形（平方）——包含 4 条等长边和 4 个直角的平面形状；或者也可描述为数字自身相乘的结果。

方阵——矩阵包含相同的行数和列数。

平方根——平方函数 x^2 的逆函数，当 x 的平方根与其自身相乘时，其结果为 x。

稳定策略——在游戏中，玩家的一组策略集。若独立玩家改变策略，则游戏体验缺乏应有的稳定性。

标准偏差——数值集的分布测算方案，定义为数值（与自身均值间的）均方距离的平方根。

指令——数值间彼此关联的“语句”。

静摩擦力——两个表面间彼此施加的摩擦力，但二者并未产生相对移动。另外，读者还可参考术语“动摩擦力”。

代入法——针对特定的参数值集合计算某一函数。

对角——若在点 P 与对象周边两点间绘制一个三角形，则点 P 处包含最大三角形的角度称作对角（点 P 常称作观察点）。

和——两个数值间的加法结果。

表面——空间内的一组连接点集，通常由一组连续参数加以定义。

旋转表面——针对某一函数 $f(x)$，3D 空间内的点集，且距 x 轴间的垂直距离等于任意 x 坐标处的 $f(x)$ 值。

切线——直线（或平面）于特定点处与曲线或表面接触（并未穿越）。也就是说，该点处与当前曲线具有相同的梯度。

趋于某一极限值——在数值序列 a_1，a_2,…中，存在值 x，并针对任意小值 d，可计算 N 以满足$|a_n - x| < d$（$n > N$）。

张力——对象从各端拉伸后所产生的作用力，例如绳索或弹簧。

张量——向量、矩阵和标量的广义化概念，表示为一维或多维中的数值阵列。

数据项——仅由常量值与标量组合间的乘积构成的数据元素或表达式，例如 $5xy^2$。

终极速度——经阻力或摩擦力作用后，降落物体的最大向下速度。

纹素——纹理贴图中的一点。

纹理贴图——用于描述表面与特定点光照间相互作用的图像贴图。

拓扑学——一门研究物体形状的学科，且未参考距离数据，其中仅适用连接性、孔洞、扭曲等符号型对象属性。

转矩——作用力的角度对应物，定义为作用力×与旋转轴间的垂直距离。

圆环体——一类 3D 形状，其点集可描述为：与特定圆之间的垂直距离为常数。

变换——一类特定的 4×4 矩阵，用于描述齐次坐标的仿射转换。

转换——一组点集与某一空间内另一组点集间的映射。

平移——该转换向空间内各点加入一个常向量。

转置——针对矩阵 **M**，行、列数据互换，反之亦然。

梯形——包含一组平行边的四边形。

树形结构——各节点仅包含一个父节点（除了根节点之外）以及 0 或多个子节点，且不包含环状结构。

三角函数——在直角三角形中，角度与其边长间的关联函数。

三角恒等式——三角函数间彼此关联的公式。

三角学——一门与三角形属性相关的学科。

转折点——函数图上的一点，其导数（以及表面的全部偏导数）为0，对应点可能是最大值点、最小值点、反射点或鞍点。

不可数集——包含无穷多个数值元素的集合。

单位向量——长度为1的向量。

未知项——方程或命题中的数据元素，可假设该命题为真进而求解对应值。

工具函数——当使用遗传算法时，针对搜索问题计算特定解的方法。

变量——函数中的数据元素，并可根据定义域内的数值（也称作参数）予以替换，进而返回其他值。另外，读者还可参考涉及常数、参数以及未知项的术语“通项”。

向量——两个或多个实数（或其他值）的有序集，并可分量实现加法计算，以及基于变量的乘法运算。

向量积——一类3D向量，其构成可描述为：针对其他两个向量，其值等于二者长度的乘积，并乘以二者间夹角的正弦值；对应方向则与两个向量垂直（也称作叉积）。

速度——物体在一段时间内运动后形成的向量。

顶点——形状的角点。在图中等价于一个节点。

视见体——基于特定相机的可见空间体。

视口——屏幕上的可见投影平面区域。

可见光——光线波长位于人眼可见的范围（约为10^{-7}m）。

波——一组耦合振子在不同地点间传递能量的物理现象。

波形——特定波的图形状，用于测算空间内距平衡位置的距离。

波前——一类空间表面，用于描述波运动过程中的振荡等价点。

波长——连续波前之间的距离。

重量——重力作用于物体上的作用力，且正比于其质量。

权值和——在数值集合x_1，x_2，…中，求和结果乘以一组既定值w_1，w_2，…，即$\sum_i w_i x_i$。

功——对象释放的能量，并引发另一对象的运动行为。

零和游戏——游戏中，玩家赚取分值，且在各个阶段中，全部分值为常数。

附录B　代 码 引 用

本书代码示例基于 Lingo 脚本语言编写。连同 JavaScript 语言，Lingo 脚本语言由 Adobe Director 提供支持。另外，相关伪代码所示算法包含了一些自然语言，例如“*set p to the nearest point to q*,” 或 “*find the nearest point to q*”。本附录旨在强调数据类型，Lingo 语言的特征及其在伪代码中的表达方式。

B.1　数 据 类 型

Lingo 是一类松散型的类型语言，这也意味着，用户无须确定特定变量所持有的数据类型，或特定函数返回的数据类型。另外，用户还可方便地在下列数据类型间进行转换。

- Boolean：TRUE 或 FALSE 值，特别地，二者等价于 1 和 0。
- Integer：标准的 32 位整数。
- Float：单精度浮点数。
- String：表示 ASCII 字的字符串，经指定后，还可使用双字节字符。
- Symbol：Lingo 语言特有的数据类型。从某种意义上讲，Symbol 类型与字符串的关系类似于浮点数与整数间的关系。另外，该类型常通过#号表示，并用于#positive 等标记中。Symbol 类型与字符串类型有几分相似，但不包含空格或非字母数字符号，并且对大小写不予区分。
- List：List 类型等价于其他语言中的链表类型，即包含未定义长度的数组。实际上，List 为一类对象，并包含多种方法可设置、删除和搜索数值。
- Point/Vector：两个或 3 个浮点值形成的特定列表。此类向量对象包含 getNormalized() 和 dot()等方法。除此之外，还存在其他列表对象，例如 rect、transform、rgb、time 等。
- Object：该类型包含多种数据，如前所述，Lingo 语言中的大多数类型均为对象。

下列代码体现了 Lingo 语言的松散类型特征：

```
set number to 7
if number then
  return "yes"
else
  return "no"
end if
```

上述代码将返回 “yes”。针对 if 语句和操作符，Lingo 语言可将数字视为 Boolen 类型数据。

B.2　变　　量

Lingo 语言中包含了 3 种类型的变量，如下所示：

- 局部变量。此类变量仅存在于特定函数的运行期内，包括函数参数以及函数中定义的变量。除非代码变得难以阅读，否则本书一般使用局部变量。
- 特征变量。此类变量为特定的对象所持有，各对象实例可包含自身的变量值。例如，标准的精灵对象包含 loc、width 和 height 等属性。其中，某些属性可直接设置，而另一些属性则无法执行此类操作。Lingo 语言的缺点之一是，无法在用户定义对象内创建私有或只读属性。在脚本对象中，可通过声明方式构建属性变量，且通常位于脚本开始处，例如属性 pSize。对象的属性可采用“.”语法（例如 object.propertyName），或者较为详细的语法说明予以显示（例如对象的 propertyName）。在示例代码中，前缀字母 p 用于命名属性变量，例如 pName 和 pMember。
- 全局变量。此类变量隶属于代码的全部范围，并可在任意时刻进行访问。尽管使用全局变量并非错误，但这并非是一种良好的编码方式。

B.3　操　作　符

本书仅采用基本的操作符，例如+，-，*，/以及=。另外，本书不使用递增操作符++，且不区分等号（==）与赋值操作符（=）之间的差别。下列代码均等价于 a+=b（从可读性值便捷性排列）：

```
set a to a+b
set a = a+b
a = a+b
```

附录C　希 腊 字 母

希腊字母常用于数学中，因而有必要了解其发音方式。表 C.1 列举了某些较为常用的字母，鉴于与罗马字母具有一定的相似性，因而某些字母较少使用。

表 C.1　希腊字母及其数学应用

希腊字母	英　语	应　用
α	Alpha	常用于表示一个角度
β	Beta	常用于表示一个角度
γ	Gamma	常用于表示一个角度
Δ	Delta	表示为无穷小数，常用于微积分运算中
ε	Epsilon	表示为无穷小数，常用于函数分析中
ξ	Zeta	偶尔表示为角度
η	Eta	偶尔表示为角度
θ	Theta	常用于表示一个角度
ι	Iota	较少使用
κ	Kappa	较少使用
λ	Lambda	该符号可表示为波长和比值，特别是特征值
μ	Mu	该符号具有“微小”或百万分之一的含义，常用于表示与各种材质相关的常数，例如摩擦系数

附录D 学 习 资 源

本附录包含了与数学和程序设计相关的扩展资源，其中包括通用数学、碰撞检测、迷宫以及游戏物理。

D.1 数 学

网络中存在大量的数学资源，且大部分内容为免费资源，www.khanacademy.org/网站即是其中之一。该网站由 Gates Foundation 提供部分资助，且包含了与数学相关的自学内容和视频素材，例如基础代数、线性代数和微分方程等，高质量且特征鲜明的教学素材可视为该网站的优势之一。

另外两个网站则是 MathWorld（http://mathworld.wolfram.com）和 ScienceWorld（http://scienceworld.wolfram.com），并由数学软件 Mathematica 的开发方 Wolfram 赞助。此类网站使用了用户提交机制，且颇具技术性。另外，网站中并未包含相关问题的详细解释。作为一类通用参考资源，Wikipedia（http://en.wikipedia.org）和 HowStuffWorks（www.howstuffworks.com）通常相对稳定。除此之外，读者还可访问 Math Forum（http://mathforum.org），并咨询与数学相关的问题，对应主页为“Ask Dr. Math”。

某些网站涵盖了与游戏计算相关的内容，例如 GameDev.net（www.gamedev.Net）和 GamaSutra（www.gamasutra.com），二者专注于 3D 技术且面向于高级程序员。另外一个面向游戏爱好者且难度适中的邮件列表型网站是 GDAlgorithms-List，对应网址为 http://lists.sourceforge.net/lists/listinfo/gdalgorithms-list。针对较为高级的数学内容，MIT 在其 MIT Open Courseware 项目中发布了多项课程，读者可访问 http://ocw.mit.edu/index.htm 获取相关列表。

针对基于数学知识的 ActionScript/JavaScript 程序设计，读者可阅读《*Macromedia Flash Professional 8 Game Development*》一书（由 Charles River Media 于 2006 年出版）。尽管相关软件不断更新，但这并不会影响到该书的价值。另外，多家网站也提供了较为基础的数学知识，其主要阅读群体面向少年儿童。BBC Education Web 网站便是其中之一，该网站面向 16 岁以上的人群，内容完整且精彩，对应网址为 www.bbc.co.uk/schools/16/maths.shtml。

这里，建议读者阅读《*Beginning Math and Physics for Game Programmers*》一书（Wendy Stahler 等人编写，New Riders 于 2004 年出版），该书内容相对浅显，但包含了早期的基础知识。相比而言，《*3D Math Primer for Graphics and Game Development*》一书则较为高级，该书由 Fletcher Dunn-Ian Parberry 编写，Jones& Bartlett 于 2002 年出版。《*Physics for Game Developers*》一书介绍了与物理学相关的数学知识（David Bourg 编写，O'Reilly 于 2001 年出版），该书讨论了与作用力相关的物理模拟建模方式。《*Essential Mathematics for Games and Interactive Applications*，*2nd Edition*》一书则相对高级，该书由 James M. Van Verth-Lars M. Bishop 编写，Morgan Kaufmann 于

2008 年出版。

D.2 专 业 资 源

本节主要关注特定领域的专业教材，其内容并非仅限于游戏开发，同时也是本书面向读者所强调的知识。

D.2.1 碰撞检测

关于碰撞检测，读者可参考 Christer Ericson 编写的《*Real-Time Collision Detection*》一书（Morgan Kaufmann 于 2005 年出版），该书涵盖了诸多与技术相关的话题；而 Gino Van Den Bergen 编写的《*Collision Detection in Interactive 3D Environments*》一书则以相对简约的方式阐述了同一话题（Morgan Kaufmann 于 2003 年出版）。David Eberly 编写的《*3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics，2nd Edition*》一书则视为该领域内的经典专著，并包含了丰富的专业内容。Chris Hecker 编写了与刚体动力学相关的系列教程，该教程难度适宜且兼具较好的可读性，并包含了对应的源代码，对应网址为 www.d6.com/users/checker/dynamics. htm#articles。

D.2.2 3D 引擎和几何学

在前述介绍中，3D 方面的资源占据了大部分内容，且多数均会对图形学问题予以关注。有鉴于此，《*Mathematics for 3D Game Programming and Computer Graphics，Third Edition*》一书讨论了与此相关的数学知识（Eric Lengyel 编写，Course Technology PTR 于 2011 年出版），尽管其内容范围相对狭窄。

D.2.3 游戏物理

《*The Illustrated Principles of Pool and Billiards*》一书介绍了撞球类游戏的开发规范（David Alciatore 编写，Sterling 于 2004 年出版），www.engr.colostate.edu/～dga/pool/technical_proofs/index.html 对该书提供了支持。关于更加广泛的物理内容，读者可参考 Dave Eberly 编写的《*Game Physics，2nd Edition*》一书（Morgan Kaufmann 于 2010 年出版）。

D.2.4 迷宫、搜索和人工智能

多本专著均对 AI 以及视频游戏中的相关话题进行了讨论，其中包括 Guy Lechy-Thomson 编写的《*AI and Artificial Life in Video Games*》一书（Charles River Media 于 2008 年出版）、Neil Kirby

编写的《*Introduction to AI*》一书（Course Technology PTR 于 2010 年出版）以及 Dave Mark 编写的《*Behavioral Mathematics for Game AI*》一书（Course Technology PTR 于 2009 年出版）。Mat Buckland 编写的《*Programming Game AI by Example*》一书年代稍显久远（Jones & Bartlett 于 2004 年出版），但书中内容难度适中。Ian Millington 编写的《*Artificial Intelligence for Games，2nd Edition*》则是另一部经典专著（Morgan Kaufmann 于 2009 年出版），该书最先讨论了游戏中的人工智能问题。总体而言，关于视频游戏中的各种搜索类型，网络中提供了过剩的资源。对此，读者可通过二分、字符串、调和（harmony）、贪婪算法、模式、二分树等关键字查询搜索算法。而对于算法的综合介绍，读者可参考 Cormen-Charles Leiserson-Ronald Rivest-Clifford Stein 联手编写的《*Introduction to Algorithms，Third Edition*》一书（MIT Press 于 20009 年出版），该书也是广泛使用的大学教材。关于迷宫方面的内容，建议读者阅读 *Think Labyrinth*，对应网址为 www.astrolog.org/labyrnth.htm。

附录E　练　习　答　案

鉴于篇幅问题，本书并未提供全部练习的代码清单。另外，本附录仅给出了练习的相关建议以及简要的代码片段。

读者可访问本书的辅助网站 www.coursePTR.com/downloads 获取相关章节的代码示例，对应示例采用 Lingo 语言编写。

【练习 1.1】当编写 convertBase(NumberString, Base1, Base2)函数时，较为简单的方式是编写一个函数，并将字符串从基数 1 转换为数字（虽然该方法相对低效），并于随后编写另一个函数将其从数字转换为基数 2。这里，读者可使用文中所提供的相关函数，但需要以浮点数方式考察 base()和 fromBase()函数。

【练习 1.2】浮点数。该问题较为简单，读者可尝试处理除法运算，即实现长除法的二进制版本。在各个阶段，可获得当前余数，并附以被除数的各数据位，直至大于或等于除数。此时，可减去除数并计算新余数。该过程持续进行，直至到达被除数的结尾。当使用浮点数时，可尝试编写 mantissaExponent()函数，并将数字转换为 IEEE 风格的浮点数。

【练习 2.1】文本滚动栏。关于滚动栏是否应根据文本图像的尺寸抑或其中的字符数量进行计算，原题并未给予清晰描述。也就是说，滚动栏比例是否可采取“像素/字符”或“像素/像素”方式进行计算。第二种方法则更为标准（也相对简单）：读者可首先尝试制作滚动栏，并查看其工作方式。对此，需要计算任意时刻屏幕上的可见字符数量，进而计算滚动栏的最大范围。

【练习 2.2】当构建 compoundinterest(amount, percentage, years)函数时，关键之处在于针对每一年乘以(100 + percentage)/100。

【练习 2.3】该练习体现了通过对数和计算两个数字的乘积。对此，可使用下列语句：

```
return exp(ln(a)+ln(b))
```

另外，图像元素也不可或缺，但该过程相对复杂。

【练习 3.1，3.2，3.3】求解方程。相关函数依然可视为整体概念中的部分内容。本书资源文件中包含了 substitute()函数的扩展实现方法，该函数使用了其他函数的返回值。需要注意的是，针对计算机可读形式的表达式，calculateValue()函数中的简单方法工作良好。这里的技巧是使用递归方案。例如，当简化某一表达式时，可分别简化各项，并于随后简化其子项等，直至得到相应的简化结果。

【练习 4.1】solvetriangle(triangle)函数。多数时候，此类函数可视为一类简单的、具有簿记性质的练习，且存在须单独处理的、多种数据组合方式。其中，最为简单的方式是连续使用各 rules(sine, cosine)函数直至结束。

【练习 4.2】rotatetofollow(triangle,point)函数。对此，一种最为简单的方法是使用本书后续内容定义的函数，例如 rotateVector()函数。这里，建议读者阅读第 5 章之后的内容，则相关问题

将会变得越发明晰。简而言之，此处需要计算中心位置-三角形顶点直线所形成的角度，以及中心位置至当前点间的角度，并计算二者之差。随后，还需使用 sin()和 cos()再次计算三角形的位置。

【练习 5.1】向量绘制。该练习并不存在“真正”的答案，此处建议读者进行多方尝试，并考察第 5 章中的 createA()函数。另外，有兴趣的读者还可尝试绘制 3D 图像。

【练习 5.2】calculateTrajectory(oldPosition, newPosition, speed)函数。该函数具有自解释特征，其实现过程在其他示例中也有所体现。

【练习 6.1】drawDifferentialEquation(function)函数。从第 6 章的求解代码中可看到，该过程相对复杂，这在其他函数图中也有所体现。也就是说，需要计算相应的缩放尺寸，计算轴中标记的最佳尺寸等。该函数的核心内容如下所示：

```
d=calculate2DValue(functionToDraw,pt[1], pt[2])
if d=#infinite then
  p1=0
  p2=yspacing
else if d<>#undefined then
  p=point(1.0,d)
  len=sqrt(1.0+d*d)
  p=p/len
  p1=p[1]*xspacing
  p2=p[2]*yspacing
else
  next repeat
end if
```

这里，假设读者位于某一特定点，并从该点处绘制一条直线，随后再次在该直线的端点处开始操作。其中，较为困难的部分是生成合理的直线分布状态。对此，可简单地选择多个随机起始点，并查看最终位置。

【练习 6.2】secantMethod()函数。该函数可视为本章代码的简单扩展，旨在实现快速计算，这可通过缓存多个引用值加以实现。

【练习 7.1】javelin(throwAngle, throwSpeed, time)函数。该函数的关键之处是计算各阶段的速度向量，并于随后旋转标枪以使其保持一致。需要注意的是，代码涵盖了某些简单的碰撞检测机制，进而确定标枪何时到达屏幕的边缘（尽管标枪可以驶离屏幕上方）。

【练习 7.2】aimCannon(cannonLength, muzzleSpeed, aimPoint)函数。该练习包含了某些技巧，相关计算过程相对复杂，下列内容讨论了相应的实现方式。

假设火炮的长度为 l，速度为 v，且目标为位置向量 $\mathbf{p}$ 的一点。若球体对象的发射速度表示为 v，则该对象在时刻 t 时的位置如下所示：

$$\left((l+tv)\cos\theta,(l+tv)\sin\theta-\frac{gt^2}{2}\right)$$

相关求解结果如下所示：

$$(l+tv)\cos\theta=p_1$$

$$(l+tv)\sin\theta-\frac{gt^2}{2}=p_2$$

当前，存在多种方法可对其进行求解。例如，可采取下列方式直接进行计算：替换第二个方程中的 sinθ，进而生成基于 t 的二次方程，并可通过代数方式求解；或者可采用近似解这一相对简单的方案，并通过迭代方式逼近最终结果。

另外，可定义一个函数用于测试当前目标点的高低程度（当球体的 x 位置为 p_1 时，计算其 y 位置）。当目标位置较高时，该函数应返回 1；目标位置较低时则返回−1；若击中目标或十分接近目标时，函数则返回 0。需要注意的是，若火炮直接瞄准目标位置，则通常无法击中目标——若使用该角度作为基线，并以渐进方式递增，直至获取位置稍高的发射角。随后，可采用简单的二分近似法（binary approximation method）求解。

【练习 7.3】fireCannon(massOfBall, massOfCannon, energy)函数。该函数较为直观，并使用了本章中的计算公式。

【练习 8.1】pointParallelogramCollision(pt, parr, tm)函数。本章建议使用斜转换并将平行四边形转换为标准的矩形。若期望对当前问题直接求解，则可将其描述为与 4 个独立墙面间的碰撞行为。通过计算墙面法线与粒子速度之间的点积，可获得包含潜在碰撞结果的墙面。随后，可使用常规方法确定碰撞结果。

【练习 8.2】内部碰撞。与外部碰撞相比，内部碰撞则相对简单。例如，圆形内部的矩形仅在顶点处发生碰撞；而矩形内部的圆形则不存在顶点的碰撞；矩形内部的矩形则仅在顶点处发生碰撞（若二者轴对齐）。

【练习 9.1】checkCollision()函数。该函数相对抽象且涉及相关技巧。读者将会发现，若缺乏一个通用碰撞检测环境，则碰撞测试通常难以执行。本章中的函数版本包含了大量的细节内容，读者可不断对其进行尝试。

【练习 9.2】牛顿摆。取决于简化方式，该问题的难度也不一而同，而当前目标则是使问题趋于简化。

【练习 10.1】splitPolygon(poly)函数。正如本章所建议的那样，该函数可采用递归方式加以定义。对此，可选择 3 个邻接点，检测三角形中的第三条直线是否与其他直线相交。若是，则对当前形状进行划分。

【练习 10.2】smoothNormals(collisionMap)函数。该函数出现于本章“对象的建模与碰撞”这一讨论中。在各步骤中，该函数检测碰撞图的全部像素，针对边像素，函数计算邻接像素并设置邻接像素均值法线。在大约 4 次迭代操作之后，可获得相对于当前表面的近似匹配结果。

【练习 11.1】检测落袋球。当球体对象途径袋口时，将于袋口定义的圆形形状相交。尽管缺乏应有的优雅性，但该过程相对简单。另一种方法则是检测沿落袋入口绘制的直线。

【练习 11.2】球体预览。该问题并非想象中的困难，关键之处是执行常规的碰撞检测，并辅以较大的速度值。当出现首次碰撞时，可将计算结果绘制至当前图像中。

【练习 12.1】碰撞场景。如本章所述，由于缩放问题，此类模拟通常包含有限的成功率。尽管如此，当前方案尚工作良好。

【练习 13.1】resolveCushionCollision(obj1, obj2, normal, moment, slow)函数。读者可参考本章中的“提示”内容以对该函数加以考察。

【练习 13.2】处于旋转状态的运动正方形与墙面间的碰撞问题。尽管该问题颇具技巧性，但并非不可求解。考虑到无须处理墙面两端，因而碰撞仅发生于正方形顶点和墙面之间。对此，可通过近似法获取一定的计算精度。例如，一类较为简单的计算方法可描述为：正方形顶点是否始于墙面一侧，并最终位于其他墙面上。针对一般需求，对应结果尚且令人满意。

【练习 14.1】applyFriction(velocity, topSpin, radius, mass, muK, muS, time)函数。虽然实现过程较为直观，但难点在于处理撞球游戏中的旋转问题，即上旋问题。一旦球体改变方向，与行进方向相比，该球体对象将以一个奇特的角度产生旋转，这将使问题复杂化。

【练习 15.1】火箭对象。对于该问题，读者可参考书中内容获得问题的处理方法。

【练习 16.1】弹簧。各种版本的计算方式均会产生不同程度的误差，其中，难点在于获取正确的弹性极限值，对应代码并未完全予以实现。

【练习 17.1】绘制 3D 立方体对象，并执行背面剔除操作。该问题的实现过程较为简单，也就是说，无须绘制外向法线远离观察者的表面，此类表面与相机-表面向量间存在正点积值。在线框绘制模式中，可针对逐顶点进行操作。也就是说，若共享该顶点的全部表面面向后面，则该顶点被剔除。随后，可绘制两个端点皆可见的直线。代码中的下一步则是生成 z 排序，即相对于相机，在远距离对象之前绘制近距离对象。

【练习 18.1】使用相对转换。第 18 章中对此提供了较为直观的解释。

【练习 19.1】3D 碰撞。该练习并未确定所实现的碰撞。当前，读者应习惯于采用公式方式对问题进行求解。至少，读者会发现球体和平面间的碰撞计算较为简单。

【练习 20.1】纹理贴图。存在多种方式可生成网格，但读者仅需考察与 textureCoordinates 属性相关的直线。

【练习 21.1】2D NURBS。若读者参考本章中的相关公式，则会发现该问题的实现过程十分简单。

【练习 21.2】IKapproach(chain, target)函数。该函数的关键之处在于反向求解，即首先调整中心肢体，以使其朝向目标旋转，并于随后调整邻接肢体，直至最后一块骨骼。

【练习 22.1】Complete drawBresenham()函数。读者可参考第 22 章中的相关函数，该函数可视为其简单的扩展。

【练习 22.2】车辆控制问题。物理模拟的目标之一是如何简化问题，并使其具有一定的复杂性。当前练习即体现了简洁性和复杂度之间的平衡关系。

【练习 23.1】box3DTileTopCollision()函数。该问题相对复杂，当使用非对齐盒体时尤其如此。

【练习 24.1】迷宫遍历问题。对此，可考察迷宫创建过程中的递归回溯机制。

【练习 25.1】头、尾问题。少量的方案可对某些基本问题加以改善。

【练习 25.2】群集问题。大量的资料均对该问题有所讨论，除此之外，读者还可查看某些在线资源，例如 Craig Reynolds 的个人网站，其网址为 www.red3d.com/cwr/boids。

【练习 26.1】8 皇后遗传算法。geneticAlgorithm()示例函数显示了一种处理方案，并通过“变辐射（varying radiation）”系统提升计算速度（另外，当执行此类扩展操作时，该程序还展示了程序性能的查看方式）。